REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
28-04-2006	Final	01-05-2005 to 28-04-2006

4. TITLE AND SUBTITLE
Reasoning by Augmenting a Description Logic Reasoner (Phase 1) Final Report

5a. CONTRACT NUMBER
HR0011-05-C-0094

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Peter F. Patel-Schneider

5d. PROJECT NUMBER
AO Number: T973

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Lucent Technologies
5440 Millstream Rd, Suite E200
McLeansville, NC 27301

8. PERFORMING ORGANIZATION REPORT

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 N. Fairfax, Dr
Arlington, VA 22203-1714

10. SPONSOR/MONITOR'S ACRONYM(S)
DARPA/IPTO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This is the final report on the DARPO IPTO project Reasoning by Augmenting a Description Logic Reasoner (Phase 1), contract HR0011-05-C-0094, providing information on the objectives of the program and how these objectives were addressed. The Reasoning by Augmenting a Description Logic Reasoner Project (Phase 1) was designed to provide languages and tools for reasoning about information expressed in expressive Description Logics or ontology languages similar to the W3C OWL DL Web Ontology Language. The results of the project were designs of new Description Logics and new reasoning methods and optimizations for these Description Logics as well as the FaCT++ Description Logic reasoner.

15. SUBJECT TERMS
Reasoning systems, description logics, semantic web

16. SECURITY CLASSIFICATION OF: Unclassified (U)			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	169	Peter F. Patel-Schneider
					19b. TELEPHONE NUMBER *(include area code)* 908-582-4471

Final Report

for

Reasoning by Augmenting a Description Logic Reasoner (Phase 1)

This material is based upon work supported by the
Defense Advanced Research Projects Agency
DARPA/IPTO
ARPA Order No. T973
Program Code No. 4E20
Issued by DARPA/CMO under Contract HR0011-05-C-0094

Prepared by

Peter F. Patel-Schneider
Lucent Technologies, Inc.

28 April 2006

Table of Contents

Reasoning by Augmenting a Description Logic Reasoner (Phase 1)

1 Introduction

This is the final report on the DARPO IPTO project Reasoning by Augmenting a Description Logic Reasoner (Phase 1), contract HR0011-05-C-0094, providing information on the objectives of the program and how these objectives were addressed. The Reasoning by Augmenting a Description Logic Reasoner Project (Phase 1) was designed to provide languages and tools for reasoning about information expressed in expressive Description Logics or ontology languages similar to the W3C OWL DL Web Ontology Language. The results of the project were designs of new Description Logics and new reasoning methods and optimizations for these Description Logics as well as the FaCT++ Description Logic reasoner.

2 Task Objectives and Technical Problems

The first objective of the project was the design of new reasoning methods and optimizations for reasoning in expressive representation languages (i.e., Description Logics) similar to the W3C OWL DL Web Ontology Language (http://www.w3.org/TR/owl-ref), including optimized reasoning methods for hybrid datatype extensions of Description Logics. The project also aimed to catalogue known relevant methods and optimizations and to analyze them for suitability.

A closely related objective was the implementation of an optimized inference engine for the expressive Description Logic and the hybrid extensions. The inference engine was to incorporate at least the optimizations in current highly optimized systems, such as DLP, FaCT, and RACER, that are applicable to expressive Description Logics, as well as new optimizations designed for these languages. The performance of the inference engine was to be empirically analyzed on a variety of inputs, including existing ontologies.

The third objective of the project was the design and implementation of an interface to control the inference engine and examine its performance. This interface was to allow the support of larger reasoning tasks, both by tuning the inference engine for a particular task and by allowing a larger system to inspect the inference engine's deliberations.

3 General Methods and Technical Results

An analysis of Description Logic constructs similar to those in the W3C OWL DL Web Ontology Language was performed. Several additions to OWL DL were identified as useful for the purposes of the project.

One of these extensions involves the addition of a form of rules to OWL DL. This extension is documented in the paper "OWL rules: A proposal and prototype implementation" by Ian Horrocks, Peter F. Patel-Schneider, Sean Bechhofer, and Dmitry Tsarkov, Journal of Web Semantics, 3(1):23-40, July 2005. This paper is available at http://www.websemanticsjournal.org/ps/pub/2005-2 and as Appendix A.

A second extension involves the addition of datatypes and datatype predicates to OWL DL. This extension is documented in the paper "Introducing Customised Datatypes and Datatype Predicates into OWL" by Jeff Z. Pan and Ian Horrocks, 2005 OWL: Experiences and Directions Workshop, Galway, Ireland, November 2005. This paper is available at http://www.mindswap.org/2005/OWLWorkshop/sub10.pdf and as Appendix B.

A third extension involved the addition of operators that allow the construction of complex relationships, such as construction of the uncle relationship from the parent and brother relationships. An early version of this extension is documented in the paper "The Irresistible SRIQ" by Ian Horrocks, Oliver Kutz, and Ulrike Sattler, 2005 OWL: Experiences and Directions Workshop, Galway, Ireland, November 2005. This paper is available at http://www.mindswap.org/2005/OWLWorkshop/sub20.pdf and as Appendix C. The final version of this extension is documented in the paper “The Even More Irresistible SROIQ” by Ian Horrocks, Oliver Kutz, and Ulrike Sattler, in Proceedings of the Tenth International Conference on Knowledge Representation and Reasoning, June 2006. A version of this paper is attached as Appendix L.

A report on the use of Description Logics similar to OWL DL in support of actual reasoning tasks is contained in the paper "Description Logics in Ontology Applications" by Ian Horrocks, Proceedings of the 9th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2005), Koblenz, Germany, September 2005, Lecture Notes in Artificial Intelligence number 3702, pages 2-13, Springer, 2005. This paper is available at http://www.cs.man.ac.uk/~horrocks/Publications/download/2005/Horr05b.pdf and as Appendix D.

Portions of these extensions were incorporated into a proposal for an extension to the W3C OWL DL Web Ontology Language. This extension, called OWL 1.1, is documented in the report "OWL 1.1 Web Ontology Language Syntax" by Peter F. Patel-Schneider, Bell Labs Research,

Lucent Technologies, 12 January 2006. This report is available at http://www-db.research.bell-labs.com/user/pfps/owl/syntax.html and as Appendix E. The overall OWL 1.1 effort has a home page at http://owl1-1.cs.manchester.ac.uk/.

An analysis of general techniques for optimizing Description Logic reasoning was performed. Existing techniques were categorized being suitable for use in expressive Description Logics or not. Several new techniques were devised and also deemed suitable for use in expressive Description Logics.

The optimizations are documented in three papers: "Optimised Classification for Taxonomic Knowledge Bases" by Dmitry Tsarkov and Ian Horrocks, in Proceedings of the 2005 Description Logic Workshop (DL-2005), Edinburgh, Scotland, July 2005 (available at http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-147/39-TsarHorr.pdf and as Appendix F), "Ordering heuristics for description logic reasoning" by Dmitry Tsarkov and Ian Horrocks, in Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, Scotland, August 2005 (available at http://www.cs.man.ac.uk/~horrocks/Publications/download/2005/TsHo05a.pdf and as Appendix G), and "Optimised Terminological Reasoning for Expressive Description Logics" by Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider, which has been submitted to the Journal of Automated Reasoning. (The submitted version appears as Appendix H.)

A new Description Logic system was implemented in C++. This system, FaCT++, incorporates many existing optimizations, identified during the project, as well as some new optimizations, also identified during the project. FaCT++ can reason over several expressive Description Logics, including SHOIQ (and thus is a reasoner for the OWL DL).

An overall description of FaCT++ can be found in "System Description: FaCT++1.0" by Dmitry Tsarkov and Ian Horrocks. This document is enclosed with electronic forms of this report as Appendix I.

The code and documentation for FaCT++ can be downloaded from http://owl.man.ac.uk/factplusplus/. A copy of the source code and documentation as of 21 April 2006 appears as Appendix M.

Two analyses of the performance of Description Logic systems on existing ontologies were performed, one with previously-existing systems and one including the new Description Logic system. The analyses are reported on in two papers: "Benchmarking DL Reasoners Using Realistic Ontologies" by Zhengxiang Pan, in OWL: Experiences and Directions workshop, Galway, Ireland, November 2005 (available at http://www.mindswap.org/2005/OWLWorkshop/sub6.pdf and as Appendix J) and "Automated

Benchmarking of Description Logic Reasoners" by Tom Gardiner, Ian Horrocks, and Dmitry Tsarkov, in Proceedings of the 2006 Description Logic Workshop (DL-2006), June 2006 (enclosed as Appendix K).

An analysis of the existing DIG 1.1 interface to Description Logic systems was performed. A proposal for a new interface to Description Logic systems, called DIG 2.0, was underway as of the end of the project. DIG 2.0 is the result of efforts by an international group of researchers, including from this project. Information on DIG 2.0 can be found at http://homepages.cs.manchester.ac.uk/~seanb/dig/.

4 Important Findings and Conclusions

The project resulted in two different kinds of findings.

The first finding relates to extensions of expressive Description Logics. It was found that a small number of extensions (composition of relationships, inclusions of composed relationships, and punning) dramatically improve the acceptability of the W3C OWL DL Web Ontology Language. These extensions have been incorporated into the OWL 1.1 language, which is being implemented by several groups worldwide.

The second findings relate to implementation of reasoners for expressive Description Logics. The FaCT++ Description Logic reasoner constructed during the project is competitive with existing systems and adequate for a considerable number of reasoning tasks, a result that was somewhat unexpected. However no existing Description Logic reasoner is adequate for many other reasoning tasks, as expected, indicating that more research needs to be done in further optimization of such reasoners.

5 Significant Developments

The most significant developments of the project are:

1.a new analysis of optimizations for reasoning in expressive Description Logics, including new optimizations;

2.the FaCT++ Description Logic system, a new high-performance Description Logic system able to reason over the W3C OWL DL Web Ontology Language; and

3.the OWL 1.1 extension to the W3C OWL DL Web Ontology Language.

6 Special Comments

The work in this project was greatly assisted by research interactions with the worldwide community interest in Description Logics and ontologies. In particular, the OWL 1.1 and DIG 2.0 efforts were the result of collaborations between geographically distributed researchers. The results of such interactions are of the highest importance in the development of languages and systems for ontology reasoning. Only in this way can general consensus on methods be achieved and this consensus is needed for widespread adoption.

7 Implications for Further Research

It turned out that getting to state of the art in Description Logic reasoners was not hard. The FaCT++ reasoner developed for Phase 1 of the project was only supposed to be a testbed for further work on optimization, but it turned out to be competitive with existing high-performance reasoners. However, although FaCT++ and other existing high-performance Description Logic reasoners are now suitable for many tasks, there are many other tasks for which they are not fast enough. Therefore more research and implementation on optimizations for reasoning in Description Logics and ontologies are needed.

During the course of the project it was evident that the existing state of the Semantic Web is in flux and that its overall design still needs work. The inadequate foundation of the Semantic Web resulted in considerable work being done in the project to get around its limitations. These problems are currently affecting the W3C Rule Interchange Format working group. Research needs to be performed and consensus achieved in this important resource.

As stated above it was important to gather international groups to perform research related to some of the goals of the project. In this project the groups (the DIG group and the OWL 1.1 group) were ad hoc groups of researchers interested in research related to the goals of the project. There are still quite a number of areas where further collaborative research related to the goals of the project is needed, including future work by the DIG and OWL 1.1 groups but also work on rules and foundations of the Semantic Web. DARPA has played a major role in this arena before, notably with the DAML project's development of the DAML+OIL web ontology language---an important precursor to the W3C OWL Web Ontology Language.

DARPA can and should continue to play an important role in future work in this vein, as it is one of the very few institutions in the United States that funds large, collaborative projects. DARPA can and should support future collaborative work on languages for Semantic Web and for agent communication. Such work is of interest to DARPA's customers, as evidenced by the upcoming

Language for Intelligent Machines workshop sponsored by the US Army Research Office. Standards bodies (e.g., W3C, etc.) do not have efforts that attract nearly the same caliber of researchers, but can serve to finalize and disseminate the results of such DARPA projects.

8 List of Abbreviations and Acronyms

DAML – DARPA Agent Markup Language
DARPA – Defense Advanced Research Projects Agency
DL – Description Logic
DLP – Description Logic Processor
FaCT – Fast Classification of Terminologies
IPTO – Information Processing Technology Office
OIL – Ontology Inference Language
OWL – W3C Web Ontology Language
RACER – Reasoner for Aboxes and Concept Expressions Renamed
SRIQ – SRIQ Description Logic
SROIQ – SROIQ Description Logic
W3C – World Wide Web Consortium

9 Appendices

Appendix A:

"OWL rules: A proposal and prototype implementation" by Ian Horrocks, Peter F. Patel-Schneider, Sean Bechhofer, and Dmitry Tsarkov, Journal of Web Semantics, 3(1):23-40, July 2005. Enclosed with the full electronic version of this report as the file jws-rules.pdf.

OWL Rules: A Proposal and Prototype Implementation

Ian Horrocks Peter F. Patel-Schneider Sean Bechhofer
Dmitry Tsarkov

February 28, 2005

Abstract

Although the OWL Web Ontology Language adds considerable expressive power to the Semantic Web it does have expressive limitations, particularly with respect to what can be said about properties. We present SWRL (the Semantic Web Rules Language), a Horn clause rules extension to OWL that overcomes many of these limitations. SWRL extends OWL in a syntactically and semantically coherent manner: the basic syntax for SWRL rules is an extension of the abstract syntax for OWL DL and OWL Lite; SWRL rules are given formal meaning via an extension of the OWL DL model-theoretic semantics; SWRL rules are given an XML syntax based on the OWL XML presentation syntax; and a mapping from SWRL rules to RDF graphs is given based on the OWL RDF/XML exchange syntax. We discuss the expressive power of SWRL, showing that the ontology consistency problem is undecidable, provide several examples of SWRL usage, and discuss a prototype implementation of reasoning support for SWRL.

1 Introduction

The OWL Web Ontology Language [47] adds considerable expressive power to the Semantic Web. However, for a variety of reasons (see `http://lists.w3.org/Archives/Public/www-webont-wg/` and [20]), including retaining the decidability of key inference problems in OWL DL and OWL Lite, OWL has expressive limitations. These restrictions can be onerous in some application domains, for example in describing web services, where it may be necessary to relate inputs and outputs of composite processes to the inputs and outputs of their component processes [51], or in medical informatics, where it may be necessary to transfer characteristics across partitive properties [39].

Many of the limitations of OWL stem from the fact that, while the language includes a relatively rich set of class constructors, the language provided for talking about properties is much weaker. In particular, there is no composition constructor, so it is impossible to capture relationships between a composite property and another (possibly composite) property. The standard example here is the obvious relationship between the composition of the "parent" and "brother" properties and the "uncle" property.

One way to address this problem would be to extend OWL with a more powerful language for describing properties. For example, a decidable extension of the description logics underlying OWL DL to include the use of composition in subproperty axioms has already been investigated [22, 23]. In order to maintain decidability, however,

the usage of the constructor is limited to axioms of the form $P \circ Q \sqsubseteq P$, i.e., axioms asserting that the composition of two properties is a subproperty of one of the composed properties. This means that complex relationships between composed properties cannot be captured—in fact even the relatively simple "uncle" example cannot not be captured (because "uncle" is not one of "parent" or "brother").

An alternative way to overcome some of the expressive restrictions of OWL would be to extend it with some form of "rules language". In fact adding rules to description logic based knowledge representation languages is far from being a new idea. Several early description logic systems, e.g., Classic [38, 8], included a rule language component. In these systems, however, rules were given a weaker semantic treatment than axioms asserting sub- and super-class relationships; they were only applied to individuals, and did not affect class based inferences such as the computation of the class hierarchy. More recently, the CARIN system integrated rules with a description logic in such a way that sound and complete reasoning was still possible [28]. This could only be achieved, however, by using a rather weak description logic (*much* weaker than OWL), and by placing severe syntactic restrictions on the occurrence of description logic terms in the (heads of) rules. Similarly, the DLP language proposed in [14] is based on the intersection of a description logic with horn clause rules; the result is obviously a decidable language, but one that is necessarily less expressive than either the description logic or rules language from which it is formed.

In this paper we show how a simple form of Horn-style rules can be added to the OWL language in a syntactically and semantically coherent manner, the basic idea being to add such rules as a new kind of axiom in OWL DL. We show (in Section 3) how the OWL abstract syntax in the OWL Semantics and Abstract Syntax document [37] can be extended to provide a formal syntax for these rules, and (in Section 4) how the direct OWL model-theoretic semantics for OWL DL can be extended to provide a formal meaning for OWL ontologies including rules written in this abstract syntax. We will also show (in Section 5) how OWL's XML presentation syntax can be modified to deal with the proposed rules.

The extended language was originally called ORL (the OWL Rules Language), but is now much better known as SWRL (the Semantic Web Rules Language), a name that was coined when the Joint US/EU ad hoc Agent Markup Language Committee[1] developed a W3C members submission based on ORL.[2] Although SWRL includes some additional features (mainly related to datatypes and predicates) and has some minor syntactic differences, we will refer to the language described here as SWRL.

SWRL is considerably more powerful than either OWL DL or Horn rules alone. We will show (in Section 6) that the key inference problems (e.g., ontology consistency) for SWRL are undecidable, and (in Section 7) provide examples that utilise the power of the combined languages.

In Section 8 we show how OWL's RDF syntax can be extended to deal with rules, and in Sections 9 and 10 we discuss how reasoning support for SWRL might be provided. Finally (in Section 11), we summarise the main features of the SWRL proposal and suggest some directions for future work.

[1] http://www.daml.org/committee/

[2] http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

2 Overview

The basic idea of the proposal is to extend OWL DL with a form of rules while maintaining maximum backwards compatibility with OWL's existing syntax and semantics. To this end, we add a new kind of axiom to OWL DL, namely Horn clause rules, extending the OWL abstract syntax and the direct model-theoretic semantics for OWL DL [37] to provide a formal semantics and syntax for OWL ontologies including such rules.

The proposed rules are of the form of an implication between an antecedent (body) and consequent (head). The informal meaning of a rule can be read as: whenever (and however) the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

Both the antecedent (body) and consequent (head) of a rule consist of zero or more atoms. Atoms can be of the form C(x), P(x,y), sameAs(x,y) or differentFrom(x,y), where C is an OWL DL description, P is an OWL property, and x,y are either variables, OWL individuals or OWL data values. Atoms are satisfied in extended interpretations (to take care of variables) in the usual model-theoretic way, i.e., the extended interpretation maps the variables to domain elements in a way that satisfies the description, property, sameAs, or differentFrom, just as in the regular OWL model theory.

Multiple atoms in an antecedent are treated as a conjunction. An empty antecedent is thus treated as trivially true (i.e. satisfied by every interpretation), so the consequent must also be satisfied by every interpretation.

Multiple atoms in a consequent are treated as separate consequences, i.e., they must all be satisfied. In keeping with the usual treatment in rules, an empty consequent is treated as trivially false (i.e., not satisfied by any extended interpretation). Such rules are satisfied if and only if the antecedent is not satisfied by any extended interpretation. Note that rules with multiple atoms in the consequent could easily be rewritten (by applying standard rules of distributivity) into multiple rules each with an atomic consequent.

It is easy to see that OWL DL becomes undecidable when extended in this way as rules can be used to simulate role value maps [46] and make it easy to encode known undecidable problems as a SWRL ontology consistency problem (see Section 6).

3 Abstract Syntax

The syntax for SWRL in this section abstracts from any exchange syntax for OWL and thus facilitates access to and evaluation of the language. This syntax extends the abstract syntax of OWL described in the OWL Semantics and Abstract Syntax document [37].

Like the OWL abstract syntax, we will specify the abstract syntax for rules by means of a version of Extended BNF, very similar to the Extended BNF notation used for XML [52]. In this notation, terminals are quoted; non-terminals are not quoted. Alternatives are either separated by vertical bars (|) or are given in different productions. Components that can occur at most once are enclosed in square brackets ([...]); components that can occur any number of times (including zero) are enclosed in braces ({...}). Whitespace is ignored in the productions given here.

Names in the abstract syntax are RDF URI references [27]. These names may be abbreviated into qualified names, using one of the following namespace names:

rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
rdfs	`http://www.w3.org/2000/01/rdf-schema#`
xsd	`http://www.w3.org/2001/XMLSchema#`
owl	`http://www.w3.org/2002/07/owl#`

The meaning of each construct in the abstract syntax for rules is informally described when it is introduced. The formal meaning of these constructs is given in Section 4 via an extension of the OWL DL model-theoretic semantics [37].

3.1 Rules

From the OWL Semantics and Abstract Syntax document [37], an OWL ontology in the abstract syntax contains a sequence of annotations, axioms, and facts. Axioms may be of various kinds, for example, subClass axioms and equivalentClass axioms. This proposal extends axioms to also allow rule axioms, by adding the production:

axiom ::= rule

Thus a SWRL ontology could contain a mixture of rules and other OWL DL constructs, including ontology annotations, axioms about classes and properties, and facts about OWL individuals, as well as the rules themselves.

A rule axiom consists of an antecedent (body) and a consequent (head), each of which consists of a (possibly empty) set of atoms. Just as for class and property axioms, rule axioms can also have annotations. These annotations can be used for several purposes, including giving a label to the rule by using the rdfs:label annotation property.

rule ::= 'Implies('{annotation} antecedent consequent')'
antecedent ::= 'Antecedent('{atom}')'
consequent ::= 'Consequent('{atom}')'

Informally, a rule may be read as meaning that if the antecedent holds (is "true"), then the consequent must also hold. An empty antecedent is treated as trivially holding (true), and an empty consequent is treated as trivially not holding (false). Non-empty antecedents and consequents hold iff all of their constituent atoms hold. As mentioned above, rules with multiple consequents could easily be rewritten (using standard rules of distributivity) into multiple rules each with a single atomic consequent.

Atoms in rules can be of the form C(x), P(x,y), Q(x,z), sameAs(x,y) or differentFrom(x,y), where C is an OWL DL description, P is an OWL DL *individual-valued* Property, Q is an OWL DL *data-valued* Property, x,y are either variables or OWL individuals, and z is either a variable or an OWL data value. In the context of OWL Lite, descriptions in atoms of the form C(x) may be restricted to class names.

atom ::= description '(' i-object ')'
| individualvaluedPropertyID '(' i-object i-object ')'
| datavaluedPropertyID '(' i-object d-object ')'
| sameAs '(' i-object i-object ')'
| differentFrom '(' i-object i-object ')'

Informally, an atom C(x) holds if x is an instance of the class description C, an atom P(x,y) (resp. Q(x,z)) holds if x is related to y (z) by property P (Q), an atom sameAs(x,y) holds if x is interpreted as the same object as y, and an atom differentFrom(x,y) holds if x and y are interpreted as different objects.

Atoms may refer to individuals, data literals, individual variables or data variables. Variables are treated as universally quantified, with their scope limited to a given rule. As usual, only variables that occur in the antecedent of a rule may occur in the consequent (a condition usually referred to as "safety").

i-object ::= i-variable | individualID
d-object ::= d-variable | dataLiteral

i-variable ::= 'I-variable(' URIreference ')'
d-variable ::= 'D-variable(' URIreference ')'

3.2 Human Readable Syntax

While the abstract Extended BNF syntax is consistent with the OWL specification, and is useful for defining XML and RDF serialisations, it is rather verbose and not particularly easy to read. In the following we will, therefore, often use a relatively informal "human readable" form similar to that used in many published works on rules.

In this syntax, a rule has the form:

$$\mathsf{antecedent} \rightarrow \mathsf{consequent},$$

where both antecedent and consequent are conjunctions of atoms written $a_1 \wedge \ldots \wedge a_n$. Variables are indicated using the standard convention of prefixing them with a question mark (e.g., $?x$). Using this syntax, a rule asserting that the composition of parent and brother properties implies the uncle property would be written:

$$\mathsf{parent}(?a, ?b) \wedge \mathsf{brother}(?b, ?c) \rightarrow \mathsf{uncle}(?a, ?c). \quad (1)$$

If John has Mary as a parent and Mary has Bill has a brother, then this rule requires that John has Bill as an uncle. Using the abstract syntax described in Section 3.1, this rule would have been written as:

```
Implies(Antecedent(parent(I-variable(a) I-variable(b))
                   brother(I-variable(b) I-variable(c)))
        Consequent(uncle(I-variable(a) I-variable(c)))).
```

4 Direct Model-Theoretic Semantics

The model-theoretic semantics for SWRL is a straightforward extension of the semantics for OWL DL given in [37]. The basic idea is that we define *bindings*—extensions of OWL interpretations that also map variables to elements of the domain in the usual manner. A rule is satisfied by an interpretation iff every binding that satisfies the antecedent also satisfies the consequent. The semantic conditions relating to axioms and ontologies are unchanged, so an interpretation satisfies an ontology iff it satisfies every axiom (including rules) and fact in the ontology.

4.1 Interpreting Rules

From the OWL Semantics and Abstract Syntax document [37] we recall that an abstract OWL interpretation is a tuple of the form

$$\mathcal{I} = \langle R, EC, ER, L, S, LV \rangle,$$

where R is a set of resources, $LV \subseteq R$ is a set of literal values, EC is a mapping from classes and datatypes to subsets of R and LV respectively, ER is a mapping from properties to binary relations on R, L is a mapping from typed literals to elements of LV, and S is a mapping from individual names to elements of $EC(\mathsf{owl:Thing})$.

Given an abstract OWL interpretation $\mathcal{I}$, a binding $B(\mathcal{I})$ is an abstract OWL interpretation that extends $\mathcal{I}$ such that S maps i-variables to elements of $EC(\mathsf{owl:Thing})$ and L maps d-variables to elements of LV respectively. An atom is satisfied by a binding $B(\mathcal{I})$ under the conditions given in Table 1, where C is an OWL DL description, P is an OWL DL *individual-valued* Property, Q is an OWL DL *data-valued* Property, x, y are variables or OWL individuals, and z is a variable or an OWL data value.

Atom	Condition on Interpretation
$C(x)$	$S(x) \in EC(C)$
$P(x, y)$	$\langle S(x), S(y)\rangle \in ER(P)$
$Q(x, z)$	$\langle S(x), L(z)\rangle \in ER(Q)$
sameAs(x, y)	$S(x) = S(y)$
differentFrom(x, y)	$S(x) \neq S(y)$

Table 1: Interpretation Conditions

A binding $B(\mathcal{I})$ satisfies an antecedent A iff A is empty or $B(\mathcal{I})$ satisfies every atom in A. A binding $B(\mathcal{I})$ satisfies a consequent C iff C is not empty and $B(\mathcal{I})$ satisfies every atom in C. A rule is satisfied by an interpretation $\mathcal{I}$ iff for every binding B such that $B(\mathcal{I})$ satisfies the antecedent, $B(\mathcal{I})$ also satisfies the consequent.

The semantic conditions relating to axioms and ontologies are unchanged. In particular, an interpretation satisfies an ontology iff it satisfies every axiom (including rules) and fact in the ontology; an ontology is consistent iff it is satisfied by at least one interpretation; an ontology O_2 is entailed by an ontology O_1 iff every interpretation that satisfies O_1 also satisfies O_2.

4.2 Example

Consider, for example, the "uncle" rule (1) from Section 3.2. Assuming that parent, brother and uncle are *individualvaluedPropertyID*s, then given an interpretation $\mathcal{I} = \langle R, EC, ER, L, S, LV\rangle$, a binding $B(\mathcal{I})$ extends S to map the variables $?a$, $?b$, and $?c$ to elements of $EC(\mathsf{owl:Thing})$; we will use a, b, and c respectively to denote these elements. The antecedent of the rule is satisfied by $B(\mathcal{I})$ iff $(a, b) \in ER(\mathsf{parent})$ and $(b, c) \in ER(\mathsf{brother})$. The consequent of the rule is satisfied by $B(\mathcal{I})$ iff $(a, c) \in ER(\mathsf{uncle})$. Thus the rule is satisfied by $\mathcal{I}$ iff for every binding $B(\mathcal{I})$ such that $(a, b) \in ER(\mathsf{parent})$ and $(b, c) \in ER(\mathsf{brother})$, then it is also the case that $(a, c) \in ER(\mathsf{uncle})$, i.e.:

$$\forall a, b, c \in EC(\mathsf{owl:Thing}).$$
$$((a, b) \in ER(\mathsf{parent}) \wedge (b, c) \in ER(\mathsf{brother})) \rightarrow (a, c) \in ER(\mathsf{uncle})$$

5 XML Concrete Syntax

Many possible XML encodings could be imagined, but the most obvious solution is to extend the existing OWL Web Ontology Language XML Presentation Syntax [17],

which can be straightforwardly modified to deal with SWRL.[3] This has several advantages:

- arbitrary OWL classes (e.g., descriptions) can be used as predicates in rules;
- rules and ontology axioms can be freely mixed;
- the existing XSLT stylesheet[4] can easily be extended to provide a mapping to RDF graphs that extends the OWL RDF/XML exchange syntax (see Section 8).

In the first place, the ontology root element is extended so that ontologies can include rule axioms and variable declarations as well as OWL axioms, import statements etc. We then simply need to add the relevant syntax for variables and rules. In this paper we use the unspecified `owlr` namespace prefix for the newly introduced syntax (the `owlx` namespace prefix, which should be treated as being bound to `http://www.w3.org/2003/05/owl-xml`, is used for the existing OWL XML syntax). In practice, the `owlr` prefix would have to be bound to some appropriate namespace name (e.g., the OWL namespace name, the OWL XML namespace name, or some new namespace name).

Variable declarations are statements about variables, indicating that the given URI is to be used as a variable, and (optionally) adding any annotations. For example:

```
<owlr:Variable owlr:name="x1" />,
```

states that the URI x1 (in the current namespace) is to be treated as a variable.

Rule axioms are similar to OWL SubClassOf axioms, except they have owlr:Rule as their element name. Like SubClassOf and other axioms they may include annotations. Rule axioms have an antecedent (owlr:antecedent) component and a consequent (owlr:consequent) component. The antecedent and consequent of a rule are both lists of atoms and are read as the conjunction of the component atoms. Atoms can be formed from unary predicates (classes), binary predicates (properties), equalities or inequalities.

Class atoms consist of a description and either an individual name or a variable name, where the description in a class atom may be a class name, or may be a complex description using boolean combinations, restrictions, etc. For example,[5]

```
<owlr:classAtom>
  <owlx:Class owlx:name="Person" />
  <owlr:Variable owlr:name="x1" />
</owlr:classAtom>
```

is a class atom using a class name (#Person), and

```
<owlr:classAtom>
  <owlx:IntersectionOf>
    <owlx:Class owlx:name="Person" />
    <owlx:ObjectRestriction
        owlx:property="hasParent">
      <owlx:someValuesFrom
```

[3] The syntax used in the W3C Member Submission was changed slightly in order to make it more compatible with RuleML (see `http://www.ruleml.org/`).

[4] `http://www.w3.org/TR/owl-xmlsyntax/owlxml2rdf.xsl`

[5] Note that we use the `owlx` namespace prefix for the names used in examples.

```
            owlx:class="Physician" />
        </owlx:ObjectRestriction>
      </owlx:IntersectionOf>
      <owlr:Variable owlr:name="x2" />
    </owlr:classAtom>
```

is a class atom using a complex description representing Persons having at least one parent who is a Physician.

Property atoms consist of a property name and two elements that can be individual names, variable names or data values (as OWL does not support complex property descriptions, a property atom takes only a property name). Note that in the case where the second element is an individual name the property must be an *individual-valued* Property, and in the case where the second element is a data value the property must be a *data-valued* Property. For example:

```
<owlr:individualPropertyAtom
                    owlx:property="hasParent">
  <owlr:Variable owlr:name="x1" />
  <owlx:Individual owlx:name="John" />
</owlr:individualPropertyAtom>
```

is a property atom using an *individual-valued* Property (the second element is an individual), and

```
<owlr:datavaluedPropertyAtom owlr:property="grade">
  <owlr:Variable owlr:name="x1" />
  <owlx:DataValue
    rdf:datatype="&xsd;integer">4</owlx:DataValue>
</owlr:datavaluedPropertyAtom>
```

is a property atom using a *data-valued* Property (the second element is a data value, in this case an integer).

Finally, same (different) individual atoms assert equality (inequality) between sets of individual and variable names. Note that (in)equalities can be asserted between arbitrary combinations of variable names and individual names. For example:

```
<owlr:sameIndividualAtom>
  <owlr:Variable owlr:name="x1" />
  <owlr:Variable owlr:name="x2" />
  <owlx:Individual owlx:name="Clinton" />
  <owlx:Individual owlx:name="Bill_Clinton" />
</owlr:sameIndividualAtom>
```

asserts that the variables $x1$, $x2$ and the individual names Clinton and Bill_Clinton all refer to the same individual.

5.1 Example

The example rule from Section 3.2 can be written in the XML concrete syntax for rules as

```
<owlx:Rule>
  <owlr:antecedent>
```

```
    <owlr:individualPropertyAtom
                          owlr:property="parent">
      <owlr:Variable  owlr:name="a"  />
      <owlr:Variable  owlr:name="b"  />
    </owlr:individualPropertyAtom>
    <owlr:individualPropertyAtom
                          owlr:property="brother">
      <owlr:Variable  owlr:name="b"  />
      <owlr:Variable  owlr:name="c"  />
    </owlr:individualPropertyAtom>
  </owlr:antecedent>
  <owlr:consequent>
    <owlr:individualPropertyAtom
                          owlr:property="uncle">
      <owlr:Variable  owlr:name="a"  />
      <owlr:Variable  owlr:name="c"  />
    </owlr:individualPropertyAtom>
  </owlr:consequent>
</owlr:Rule>
```

6 The Power of Rules

In OWL, the only relationship that can be asserted between properties is subsumption between atomic property names, e.g., asserting that hasFather is a subPropertyOf hasParent. In Section 3.2 we have already seen how a rule can be used to assert more complex relationships between properties. While this increased expressive power is clearly very useful, it is easy to show that it leads to the undecidability of key inference problems, in particular ontology consistency.

For extensions of languages such as OWL DL, the undecidability of the consistency problem is often proved by showing that the extension makes it possible to encode a known undecidable domino problem [4] as an ontology consistency problem. In particular, it is well known that such languages only need the ability to represent an infinite 2-dimensional grid in order for consistency to become undecidable [2, 24]. With the addition of rules, such an encoding is trivial. For example, given two properties x-succ and y-succ, the rule:

$$\mathsf{x\text{-}succ}(?a, ?b) \wedge \mathsf{y\text{-}succ}(?b, ?c) \wedge \mathsf{y\text{-}succ}(?a, ?d) \wedge \mathsf{x\text{-}succ}(?d, ?e) \rightarrow \mathsf{sameAs}(?c, ?e),$$

along with the assertion that every grid node is related to exactly one other node by each of x-succ and y-succ, allows such a grid to be represented. This would be possible even without the use of the sameAs atom in the consequent—it would only be necessary to establish appropriate relationships with a "diagonal" property:

$$\mathsf{x\text{-}succ}(?a, ?b) \wedge \mathsf{y\text{-}succ}(?b, ?c) \rightarrow \mathsf{diagonal}(?a, ?c)$$
$$\mathsf{y\text{-}succ}(?a, ?d) \wedge \mathsf{x\text{-}succ}(?d, ?e) \rightarrow \mathsf{diagonal}(?a, ?e),$$

and additionally assert that every grid node is related to exactly one other node by diagonal.

The proposed form of OWL rules seem to go beyond basic Horn clauses in allowing:

- conjunctive consequents;
- class descriptions as well as class names as predicates in class atoms; and
- equalities and inequalities.

On closer examination, however, it becomes clear that most of this is simply "syntactic sugar", and does not add to the power of the language.

In the case of conjunctive consequents, it is easy to see that these could be eliminated by rewriting using standard rules of distributivity. For example, the rule

$$A \rightarrow C_1 \wedge C_2$$

is equivalent to $\neg A \vee (C_1 \wedge C_2)$ and, via distributivity, to $(\neg A \vee C_1) \wedge (\neg A \vee C_2)$, so can be rewritten as a semantically equivalent pair of rules

$$A \rightarrow C_1$$
$$A \rightarrow C_2.$$

In the case of class descriptions, it is easy to see that a description d can be eliminated from a rule simply by adding an OWL axiom that introduces a new class name and asserts that it is equivalent to d, e.g.,

$$\mathsf{EquivalentClasses}(D\ d).$$

The description can then be replaced with the name, here replacing the description d with class name D.

In the case of equality atoms, the sameAs property could easily be substituted with a "user defined" owl property called, for example, Eq. Such a property can be given the appropriate meaning using a rule of the form

$$\mathsf{Thing}(?x) \rightarrow \mathsf{Eq}(?x, ?x) \tag{2}$$

and by asserting that it is functional. It is easy to see that the interpretation of Eq corresponds to equality of elements in $EC(\mathsf{owl:Thing})$, i.e.,

$$\forall x, y \in EC(\mathsf{owl:Thing}).\langle x, y\rangle \in ER(\mathsf{Eq}) \iff x = y,$$

and that Eq could therefore be used instead of sameAs without changing the meaning of the ontology.

Proof: For the if direction, assume that for some interpretation $\mathcal{I}$ there exists an element x of $EC(\mathsf{owl:Thing})$ such that $\langle x, x\rangle \notin ER(\mathsf{Eq})$. Then a binding $B(\mathcal{I})$ could extend $\mathcal{I}$ so that S maps $?x$ to x, and rule 2 would not be satisfied by $B(\mathcal{I})$. For the only if direction, assume that for some interpretation $\mathcal{I}$ there exist elements x, y of $EC(\mathsf{owl:Thing})$ such that $\langle x, y\rangle \in ER(\mathsf{Eq})$ and $x \neq y$. From the if direction we also have that $\langle x, x\rangle \in ER(\mathsf{Eq})$, so Eq would not be functional.

The case of inequalities is slightly more complex. An owl property called, for example, Neq, can be introduced and used to capture some of the meaning of the differentFrom property by adding a rule of the form

$$\mathsf{Eq}(?x, ?y) \wedge \mathsf{Neq}(?x, ?y) \rightarrow \mathsf{Nothing}(?x). \tag{3}$$

It is easy to see that the interpretation of Neq is disjoint from the interpretation of Eq, i.e.,

$$\forall x, y \in EC(\mathsf{owl:Thing}).\langle x, y\rangle \in ER(\mathsf{Neq}) \implies x \neq y,$$

and that this leads to the implicit rule

$$\mathsf{Neq}(?x, ?y) \rightarrow \mathsf{differentFrom}(?x, ?y).$$

Proof: Assume that for some interpretation $\mathcal{I}$ there exist elements x, y of $EC(\mathsf{owl:Thing})$ such that $\langle x, y\rangle \in ER(\mathsf{Neq})$ and $\langle x, y\rangle \notin ER(\mathsf{differentFrom})$. If $\langle x, y\rangle \notin ER(\mathsf{differentFrom})$, then $x = y$ and $\langle x, y\rangle \in ER(\mathsf{Eq})$. A binding $B(\mathcal{I})$ could, therefore, extend $\mathcal{I}$ so that S maps $?x$ to x and $?y$ to y, and rule 3 would imply that $x \in EC(\mathsf{owl:Nothing})$, violating the semantic conditions on $\mathcal{I}$.

Rule 3 shows that we could eliminate differentFrom when it occurs in the consequent of a rule simply by substituting Neq. Neq does not, however, fully capture the meaning of inequality, because there could be pairs of elements in $EC(\mathsf{owl:Thing})$ that are in the extension of neither Eq nor Neq, i.e., differentFrom does *not* imply Neq. As a result, we cannot use Neq to eliminate occurrences of differentFrom in the antecedent of a rule: in order to do so would require Neq to be equivalent to the negation of Eq.

7 Examples of SWRL

We give two further examples of SWRL that serve to illustrate some of its utility, and show how the power of SWRL goes beyond that of either OWL DL or Horn rules alone.

7.1 Transferring Characteristics

The first example is due to Guus Schreiber, and is based on ontologies used in an image annotation demo [16].

$$\mathsf{Artist}(?x) \wedge \mathsf{Style}(?y) \wedge \mathsf{artistStyle}(?x, ?y) \wedge \mathsf{creator}(?x, ?z) \rightarrow \mathsf{style/period}(?z, ?y)$$

The rule expresses the fact that, given knowledge about the Style of certain Artists (e.g., van Gogh is an Impressionist painter), we can derive the style/period of an art object from the value of the creator of the art object, where Style is a term from the Art and Architecture Thesaurus (AAT),[6] Artist is a class from the Union List of Artist Names (ULAN),[7] artistStyle is a property relating ULAN Artists to AAT Styles, and both creator and style/period are properties from the Visual Resources Association catalogue (VRA),[8] with creator being a subproperty of the Dublin Core element dc:creator.[9]

This rule would be expressed in the XML concrete syntax as follows (assuming appropriate entity declarations):

```
<owlr:Rule>
  <owlr:antecedent>
```

[6] http://www.getty.edu/research/tools/vocabulary/aat/
[7] http://www.getty.edu/research/conducting_research/vocabularies/ulan/
[8] http://www.vraweb.org/
[9] http://dublincore.org/

```
      <owlr:classAtom>
        <owlx:Class owlx:name="&ulan;Artist" />
        <owlr:Variable owlr:name="x" />
      </owlr:classAtom>
      <owlr:classAtom>
        <owlx:Class owlx:name="&aat;Style" />
        <owlr:Variable owlr:name="y" />
      </owlr:classAtom>
      <owlr:individualPropertyAtom
                owlr:property="&aatulan;artistStyle">
        <owlr:Variable owlr:name="x" />
        <owlr:Variable owlr:name="y" />
      </owlr:individualPropertyAtom>
      <owlr:individualPropertyAtom
                owlr:property="&vra;creator">
        <owlr:Variable owlr:name="x" />
        <owlr:Variable owlr:name="z" />
      </owlr:individualPropertyAtom>
    </owlr:antecedent>
    <owlr:consequent>
      <owlr:individualPropertyAtom
                owlr:property="&vra;style/period">
        <owlr:Variable owlr:name="z" />
        <owlr:Variable owlr:name="y" />
      </owlr:individualPropertyAtom>
    </owlr:consequent>
  </owlr:Rule>
```

The example is interesting because it shows how rules can be used to "transfer characteristics" from one class of individuals to another via properties other than subClassOf—in this case, the Style characteristics of an Artist (if any) are transferred (via the creator property) to the objects that he/she creates. This idiom is much used in ontologies describing complex physical systems, such as medical terminologies, where partonomies may be as important as subsumption hierarchies, and where characteristics often need to be transfered across various partitive properties [34, 41, 44]. For example, the location of a trauma should be transfered across the partOf property, so that traumas located in a partOf an anatomical structure are also located in the structure itself [39]. This could be expressed using a rule such as

$$\mathsf{Trauma}(?x) \wedge \mathsf{Location}(?y) \wedge \mathsf{isLocatedIn}(?x, ?y) \wedge \mathsf{isPartOf}(?y, ?z) \rightarrow \mathsf{isLocatedIn}(?x, ?z)$$

A similar technique could be used to transfer properties to composite processes from their component processes when describing web services.

Terminology languages designed specifically for medical terminology such as Grail [40] and SNOMED-RT [48] often allow this kind of idiom to be expressed, but it cannot be expressed in OWL (not even in OWL full). Thus this kind of rule shows one way in which SWRL goes beyond the expressive power of OWL DL.

7.2 Inferring the Existence of New Individuals

The second example is due to Mike Dean, and illustrates a scenario in which we want to express the fact that for every Airport there is a map Point that has the same location (latitude and longitude) as the Airport and that is an object of "layer" (a map DrawingLayer).[10] Moreover, this map point has the Airport as an underlyingObject and has the Airport name as its Label. Note how the expressive power of SWRL allows "existentials" to be expressed in the head of a rule—it is asserted that, for every Airport, there must exist such a map point (using an OWL someValuesFrom restriction in a class atom). In this way SWRL goes beyond the expressive power of Horn rules.

The first part of this example is background knowledge about Airports and maps expressed in OWL DL. (A few liberties have been taken with the OWL DL abstract syntax here in the interests of better readability.) In particular, it is stated that map:location and map:object are *individual-valued* Properties with inverse properties map:isLocationOf and map:isObjectOf respectively; that latitude and longitude are *data-valued* Properties; that map:Location is a class whose instances have exactly one latitude and exactly one longitude, both being of type xsd:double; that layer is an instance of map:DrawingLayer; that map is an instance of map:Map whose map:name is `"Airports"` and whose map:layer is layer; and that airport:GEC is an instance of airport-ont:Airport whose name is `"Spokane Intl"` and whose location is latitude `47.6197` and longitude `117.5336`.

```
ObjectProperty(map:location)
ObjectProperty(map:isLocationOf
   inverseOf(map:location))
ObjectProperty(map:object)
ObjectProperty(map:isObjectOf
   inverseOf(map:location))

DatatypeProperty(latitude)
DatatypeProperty(longitude)
Class(map:Location  primitive
   intersectionOf(
      restriction(latitude  allValuesFrom(xsd:double))
      restriction(latitude  minCardinality(1))
      restriction(longitude  allValuesFrom(xsd:double))
      restriction(longitude  minCardinality(1))))

Individual(layer  type(map:DrawingLayer))

Individual(map  type(map:Map)
   value(map:name  "Airports")
   value(map:layer  layer))

Individual(airport:GEC  type(airport-ont:Airport)
   value(name  "Spokane Intl")
   value(location  Individual(value(latitude  47.6197)
         value(longitude  117.5336))))
```

[10] `http://www.daml.org/2003/06/ruletests/translation-3.n3`

The first rule in the example requires that if a map:Location is the sameLocation as another location, then it has the same values for latitude and longitude.

$$\begin{aligned}&\mathsf{map{:}Location}(?maploc) \wedge \mathsf{sameLocation}(?loc, ?maploc) \wedge \\ &\quad \mathsf{latitude}(?loc, ?lat) \wedge \mathsf{longitude}(?loc, ?lon) \\ &\quad\quad \rightarrow \mathsf{latitude}(?maploc, ?lat) \wedge \mathsf{latitude}(?maploc, ?lon)\end{aligned}$$

The second rule requires that wherever an airport-ont:Airport is located, there is some map:Location that is the sameLocation as the Airport's location, and that is the location of a map:Point that is an object of the map:DrawingLayer "layer". Note that the head of the rule is an atom of the form $C(?\mathsf{loc})$, where the class C is an OWL restriction.

$$\begin{aligned}&\mathsf{airport\text{-}ont{:}Airport}(?airport) \wedge \mathsf{location}(?airport, ?loc) \wedge \\ &\quad \mathsf{latitude}(?loc, ?lat) \wedge \mathsf{longitude}(?loc, ?lon) \\ &\quad \rightarrow \ \mathsf{restriction}(\mathsf{sameLocation} \\ &\qquad \mathsf{someValuesFrom}(\\ &\qquad\quad \mathsf{intersectionOf}(\mathsf{map:Location} \\ &\qquad\quad\quad \mathsf{restriction}(\mathsf{isLocationOf} \\ &\qquad\quad\quad\quad \mathsf{someValuesFrom}(\\ &\qquad\quad\quad\quad\quad \mathsf{intersectionOf}(\mathsf{map:Point} \\ &\qquad\quad\quad\quad\quad\quad \mathsf{restriction}(\mathsf{map:isObjectOf} \\ &\qquad\quad\quad\quad\quad\quad\quad \mathsf{someValuesFrom}(\mathsf{OneOf}(\mathsf{layer}))))))))))(?loc)\end{aligned}$$

The third rule requires that the map:Point whose map:location is the map:Location of an airport-ont:Airport has the airport as a map:underlyingObject and has a map:label which is the name of the airport.

$$\begin{aligned}&\mathsf{airport\text{-}ont{:}Airport}(?airport) \wedge \mathsf{map{:}location}(?airport, ?loc) \wedge \\ &\quad \mathsf{sameLocation}(?loc, ?maploc) \wedge \mathsf{map{:}Location}(?point, ?maploc) \wedge \\ &\quad\quad \mathsf{airport\text{-}ont{:}name}(?airport, ?name) \\ &\quad\quad \rightarrow \mathsf{map{:}underlyingObject}(?point, ?airport) \wedge \\ &\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathsf{map{:}label}(?point, ?name)\end{aligned}$$

8 Mapping to RDF Graphs

It is widely assumed that the Semantic Web will be based on a hierarchy of (increasingly expressive) languages, with RDF/XML providing the syntactic and semantic foundation (see, e.g., [5]). In accordance with this design philosophy, the charter of the W3C Web Ontology Working Group (the developers of the OWL language) explicitly stated that "*The language will use the XML syntax and datatypes wherever possible, and will be designed for maximum compatibility with XML and RDF language conventions.*". In pursuance of this goal, the working group devoted a great deal of effort to developing an RDF based syntax for OWL that was also consistent with the semantics of RDF [20]. It is, therefore, worth considering how this design might be extended to encompass rules.

One rather serious problem is that, unlike OWL, rules have variables, so treating them as a semantic extension of RDF is very difficult. It is, however, still possible

to provide an RDF syntax for rules—it is just that the semantics of the resultant RDF graphs may not be an extension of the RDF Semantics [15].

A mapping to RDF/XML is most easily created as an extension to the XSLT transformation for the OWL XML Presentation syntax.[11] This would introduce RDF classes for SWRL atoms and variables, and RDF properties to link atoms to their predicates (classes and properties) and arguments (variables, individuals or data values).[12] The example rule given in Section 7.1 (that equates the style/period of art objects with the style of the artist that created them) would be mapped into RDF as follows:

```
<owlr:Variable rdf:ID="x"/>
<owlr:Variable rdf:ID="y"/>
<owlr:Variable rdf:ID="z"/>
<owlr:Rule>
  <owlr:antecedent rdf:parseType="Collection">
    <owlr:classAtom>
      <owlr:classPredicate
        rdf:resource="&ulan;Artist"/>
      <owlr:argument1 rdf:resource="#x" />
    </owlr:classAtom>
    <owlr:classAtom>
      <owlr:classPredicate
        rdf:resource="&aat;Style"/>
      <owlr:argument1 rdf:resource="#y" />
    </owlr:classAtom>
    <owlr:individualPropertyAtom>
      <owlr:propertyPredicate
        rdf:resource="&aatulan;artistStyle"/>
      <owlr:argument1 rdf:resource="#x" />
      <owlr:argument2 rdf:resource="#y" />
    </owlr:individualPropertyAtom>
    <owlr:individualPropertyAtom>
      <owlr:propertyPredicate
        rdf:resource="&vra;creator"/>
      <owlr:argument1 rdf:resource="#x" />
      <owlr:argument2 rdf:resource="#z" />
    </owlr:individualPropertyAtom>
  </owlr:antecedent>
  <owlr:consequent rdf:parseType="Collection">
    <owlr:individualPropertyAtom>
      <owlr:propertyPredicate
        rdf:resource="&vra;style/period"/>
      <owlr:argument1 rdf:resource="#z" />
      <owlr:argument2 rdf:resource="#y" />
    </owlr:individualPropertyAtom>
  </owlr:consequent>
</owlr:Rule>
```

where &ulan;, &aat;, &aatulan; and &vra; are assumed to expand into the appropriate

[11] http://www.w3.org/TR/owl-xmlsyntax/owlxml2rdf.xsl

[12] The result is similar to the RDF syntax for representing disjunction and quantifiers proposed in [30].

namespace names. Note that complex OWL classes (such as OWL restrictions) as well as class names can be used as the object of SWRL's classPredicate property.

9 Reasoning Support for SWRL

Although SWRL provides a fairly minimal rule extension to OWL, the consistency problem for SWRL ontologies is still undecidable (as we have seen in Section 6). This raises the question of how reasoning support for SWRL might be provided.

It seems likely, at least in the first instance, that many implementations will provide only partial support for SWRL. For this reason, users may want to restrict the form or expressiveness of the rules and/or axioms they employ either to fit within a tractable or decidable fragment of SWRL, or so that their SWRL ontologies can be handled by existing or interim implementations.

One possible restriction in the form of the rules is to limit antecedent and consequent classAtoms to be named classes, with OWL axioms being used to assert additional constraints on the instances of these classes (in the same document or in external OWL documents). Adhering to this format should make it easier to translate rules to or from existing (or future) rule systems, including Prolog, production rules (descended from OPS5), event-condition-action rules and SQL (where views, queries, and facts can all be seen as rules); it may also make it easier to extend existing rule based reasoners for OWL (such as Euler[13] or FOWL[14]) to handle SWRL ontologies. Further, such a restriction would maximise backwards compatibility with OWL-speaking systems that do not support SWRL. It should be pointed out, however, that there may be some incompatibility between the first order semantics of SWRL and the Herbrand model semantics of many rule based reasoners.

By further restricting the form of rules and DL axioms used in SWRL ontologies it would be possible to stay within DLP, a subset of the language that has been shown to be expressible in either OWL DL or declarative logic programs (LP) alone [14]. This would allow either OWL DL reasoners or LP reasoners to be used with such ontologies, although there may again be some incompatibility between the semantics of SWRL and those of LP reasoners.

Another obvious strategy would be to restrict the form of rules and DL axioms so that a "hybrid" system could be used to reason about the resulting ontology. This approach has been used, e.g., in the CLASSIC [38] and CARIN systems [28], where sound and complete reasoning is made possible mainly by focusing on query answering, by restricting the DL axioms to languages that are *much* weaker than OWL, by restricting the use of DL terms in rules, and/or by giving a different semantic treatment to rules.

Finally, an alternative way to provide reasoning support for SWRL would be to extend the translation of OWL into TPTP[15] implemented in the Hoolet system,[16] and use a first order prover such as Vampire to reason with the resulting first order theory [42, 54]. This technique would have several advantages: no restrictions on the form of SWRL rules or axioms would be required; the use of a first order prover would ensure that all inferences were sound with respect to SWRL's first order semantics; and the use of the TPTP syntax would make it possible to use any one of a range of state of the

[13] `http://www.agfa.com/w3c/euler/`

[14] `http://fowl.sourceforge.net`

[15] A standard syntax used by many first order theorem provers—see `http://www.tptp.org`

[16] `http://www.w3.org/2003/08/owl-systems/test-results-out`

art first order provers. A prototype based on this approach is described in the following section.

10 A Prototype SWRL Reasoner

It is well known that OWL DL corresponds to the $\mathcal{SHOIND}_n^-$ Description Logic (DL), and that, like most other DLs, $\mathcal{SHOIND}_n^-$ is a fragment of classical first-order predicate logic (FOL) [10, 19, 1]. This suggests the idea of using standard methods of automated reasoning for FOL as a mechanism for reasoning with OWL DL.

This might be done by trying to create from scratch new architectures for reasoning in FOL, which would be specialised for dealing efficiently with typical DL reasoning tasks. A much less expensive option is to use existing implementations of FOL provers, with the possibility of making adjustments that exploit the structure of DL reasoning tasks. An additional attraction of using a FO prover in this way is the fact that the translation from DL to FOL can be extended to handle SWRL, providing an implementation of a SWRL reasoner.

Here we describe our initial prototype implementation of just such a SWRL reasoner, known as Hoolet. It should be noted that this initial implementation is rather simplistic, and is only intended as a preliminary feasibility study. We will, however, discuss the issue of possible optimisations.

There have been earlier investigations of the use of FOL provers to reason with description logics. Paramasivam and Plaisted, for example, have investigated the use of FOL reasoning for DL classification [36], while Ganzinger and de Nivelle have developed decision procedures for the guarded fragment, a fragment of FOL that includes many description logics [11]. The most widely known work in this area was by Hustadt and Schmidt [26], who used the SPASS FOL prover to reason with propositional modal logics, and, via well known correspondences [45], with description logics. Their technique involved the use of a relatively complex functional translation which produces a subset of FOL for which SPASS can be tuned so as to guarantee complete reasoning. The results of this experiment were quite encouraging, with performance of the SPASS based system being comparable, in many cases, with that of state of the art DL reasoners. The tests, however, mainly concentrated on checking the satisfiability of (large) single modal logic formulae (equivalently, OWL class descriptions/DL concepts), rather than the more interesting task (in an ontology reasoning context) of checking the satisfiability of formulae w.r.t. a large theory (equivalently, an OWL ontology/DL knowledge base).

In all of the above techniques, the DL is translated into (the guarded fragment of) FOL in such a way that the prover can be used as a decision procedure for the logic—i.e., reasoning is sound, complete and terminating. Such techniques have, however, yet to be extended to the more expressive DLs that underpin Web ontology languages such as DAML+OIL and OWL DL [18], and it is not even clear if such an extension would be possible.

An alternative approach, and the one we describe here, is to use a simple "direct" translation based on the standard first order semantics of DLs (see, e.g., [1]). Using this approach, an ontology/knowledge base (a set of DL axioms), is translated into a FO theory (a set of FO axioms). A DL reasoning task w.r.t. the knowledge base (KB) is then transformed into a FO task that uses the theory. Unlike methods such as Hustadt and Schmidt's functional translation, this does not result in a decision procedure for the DL. The direct translation approach can, however, be used to provide reasoning services

(albeit without any guarantee of completeness) for the expressive DLs underlying Web ontology languages, DLs for which no effective decision procedure is currently known. Moreover, the translation approach can easily deal with language extensions such as SWRL as described here.

In recent years, a number of highly efficient FO provers have been implemented [32, 50, 43]. These provers compete annually on a set of tasks, and the results are published [9]. One of the most successful general-purpose provers has been Vampire [43], and we have chosen this prover to use in our prototype.

Vampire is a general-purpose FOL prover developed by Andrei Voronkov and Alexandre Riazanov. Given a set of first-order formulas, Vampire transforms it into an equisatisfiable set of clauses, and then tries to demonstrate inconsistency of the clause set by saturating it with ordered resolution and superposition (see [3, 33]). If the saturation process terminates without finding a refutation of the input clause set, it indicates that the clause set, and therefore the original formula set, is satisfiable, provided that the variant of the calculus used is refutationally complete and that a fair strategy[17] has been used for saturation.

The main input format of Vampire is the TPTP syntax [49] (although a parser for a subset of KIF [12] has been added recently). Using the TPTP syntax in our prototype means that it would be possible to substitute Vampire with any one of a range of state of the art first order provers.

10.1 Translation issues

Translating OWL Ontologies into FOL Axioms We will only discuss the translation from DL to FOL as the correspondence between OWL DL and $\mathcal{SHOIN}\mathcal{D}_n^-$ is well known [19]. The translation ϕ maps DL concepts C and role names R into unary and binary predicates $\phi_C(x)$ and $\phi_R(x, y)$ respectively. Complex concepts and axioms are mapped into FO formulae and axioms in the standard way [7, 1]. For example, subsumption and equivalence axioms are translated into, respectively, FO implication and equivalence (with the free variables universally quantified).

As an example, let's see a translation of a couple of concept and role axioms:

DL	FOL
$R \sqsubseteq S$	$\forall x \forall y(\phi_R(x, y) \rightarrow \phi_S(x, y))$
$C \equiv D \sqcap \exists R.(E \sqcup \forall S^-.F)$	$\forall x(\phi_C(x) \equiv \phi_D(x) \wedge \exists y(\phi_R(x, y) \wedge (\phi_E(y) \vee \forall x(\phi_S(x, y) \wedge \phi_F(x)))))$
$A \sqsubseteq \geqslant 3\, R.B$	$\forall x(\phi_A(x) \rightarrow \exists y_1 \exists y_2 \exists y_3(\phi_R(x, y_1) \wedge \phi_B(y_1) \wedge \phi_R(x, y_2) \wedge \phi_B(y_2) \wedge \phi_R(x, y_3) \wedge \phi_B(y_3) \wedge (y_1 \neq y_2) \wedge (y_2 \neq y_3) \wedge (y_1 \neq y_3))$
Transitive(T)	$\forall x \forall y \forall z(\phi_T(x, y) \wedge \phi_T(y, z) \rightarrow \phi_T(x, z))$

Simple DLs (like $\mathcal{ALC}$) can be translated into the FOL class $\mathcal{L}^2$ (the FOL fragment with no function symbols and only 2 variables), which is known to be decidable [31]. The above translations of the role inclusion axiom and concept equality axiom are, for example, in $\mathcal{L}^2$. When number restrictions are added to these DLs, they can be translated into $\mathcal{C}^2$—equivalent to $\mathcal{L}^2$ with additional "counting quantifiers"—which is also known to be decidable [13].

The FOL translation of more expressive description logics, e.g., with transitive roles ($\mathcal{SHIQ}$, OWL Lite and OWL DL) and/or complex role axioms ($\mathcal{RIQ}$ [22]),

[17] I.e., all generated clauses are eventually processed

may lead to the introduction of three or more variables.[18] The above transitivity axiom for role T is an example of this case. FOL with three variables is known to be undecidable [7].

OWL DL also provides for XML schema *datatypes* [6], equivalent to a very simple form of *concrete domains* [21]. The minimum requirement for OWL DL reasoners is that they support `xsd:integer` and `xsd:string` datatypes, where support means providing a theory of (in)equality for integer and string values [37].

Our translation encodes the required datatype theory by mapping datatypes into predicates and data values into new constants. Lexically equivalent data values are mapped to the same constant, with integers first being canonicalised in the obvious way, and axioms are added that assert inequality between all the string and integer data constants introduced. If a data value `DV` and a datatype `DT` are mapped to DV and DT respectively, and `DV` is of type `DT`, then an axiom $DT(DV)$ is also added. As the `xsd:integer` and `xsd:string` interpretation domains are disjoint, we add an axiom to that effect. Finally, we add an axiom asserting the disjointness of the datatype domain (the set of data values) and the abstract domain (the set of individuals).

In accordance with the OWL DL semantics, other "unsupported" data types are treated opaquely, i.e., data values are mapped to the same constant if they are lexically identical, but no other assumptions are made (we do *not* assume inequality if the lexical forms are not identical) [37].

Translating SWRL Rules into FOL Axioms Using the translation approach, we can easily extend the first-order translation to SWRL rules and thus provide a simple implementation of a SWRL reasoner.

As we have seen, rules in SWRL are of the form:

$$B_1, \ldots, B_m \rightarrow H_1, \ldots, H_n$$

where each of the B_i or H_j are rule *atoms*. Possible rule atoms are shown in Table 2, where C is an OWL class description, R an OWL property and i and j are either OWL individual names or SWRL variables.

Table 2: Rule Atoms

Atom	**Type**
`C(i)`	Class Atom
`R(i,j)`	Property Atom
`i==j`	Equality Atom
`i!=j`	Inequality Atom

In our prototype we have only considered a simplification of SWRL where C must be a class name (rather than arbitrary class descriptions), and R must be an object property. The first of these restrictions does not affect the expressiveness of the language, as new class names can be introduced into the ontology to represent any complex descriptions required in rules. The restriction to object properties simplifies our implementation, but the translation we describe could easily be extended to handle data valued properties.

The translation of rules exactly follows the semantics of the rules as given in Section 4. Each rule is translated as an implication, and any free variables in the rule are

[18] In some cases, the effects of transitive roles can be axiomatised in $\mathcal{C}^2$ [53].

assumed to be universally quantified. Thus a rule:

$$B_1, \ldots, B_m \rightarrow H_1, \ldots, H_n$$

is translated to an axiom:

$$\forall x_1, x_2, \ldots, x_k . T(B_1) \wedge \ldots \wedge T(B_m) \rightarrow T(H_1) \wedge \ldots \wedge T(H_n)$$

where $x_1, x_2, \ldots, x_k$ are all the variables occurring in the B_i and H_j.

Translation of atoms is trivial and is shown in Table 3. Combining this translation with the translation from OWL to FOL described above provides us with a prototype implementation of a SWRL reasoner. Given an ontology and a collection of rules relating to that ontology, we translate the ontology to FOL, and then add the FOL axioms generated by translating the rules. The resulting theory is passed to a FO prover (Vampire in our case), where it can be used for reasoning tasks such as satisfiability checking and instance checking.

Table 3: Rule Atom Translation

Atom	**Translation**
`C(i)`	$C(i)$
`R(i,j)`	$R(i,j)$
`i==j`	$i=j$
`i!=j`	$i \neq j$

10.2 Examples

As an example, we will consider a variant on the "uncle" example given in Section 3.2:

```
hasParent(?x,?y), hasSibling(?y, ?z), Male(?z)
    ⇒ hasUncle(?x,?z)
```

If our ontology additionally includes the axiom and facts (expressed here using standard DL syntax):

$$\texttt{Uncle} \equiv \exists \texttt{hasUncle}^{-}.\top$$
$$\langle \texttt{Robert}, \texttt{Paul} \rangle : \texttt{hasParent}$$
$$\langle \texttt{Paul}, \texttt{Ian} \rangle : \texttt{hasSibling}$$

then the reasoner can infer not only `hasUncle(Robert, Ian)`, but also that `Ian` is an instance of the `Uncle` class.

Another interesting aspect of the language is illustrated by the following rule:

```
Beer(?x) ⇒ Happy(Sean)
```

This expresses the fact that for any instances of the class `Beer`, `Sean` must be an instance of `Happy`. This effectively allows us to express an existential quantification over the class `Beer`: if we can prove the existence of an instance of this class, then `Sean` will be `Happy`. Note that we do not actually have to provide a name for such an instance. For example, if our ontology includes the fact:

$$\texttt{Sean} : \exists \texttt{drinks}.\texttt{Beer}$$

then the reasoner can infer that `Sean` must be `Happy` as we now know that there exists *some* instance of `Beer`—even though this instance is unnamed.

10.3 Performance and Optimisation

Our prototype works well with small examples, such as those given in Sections 7 and 10.2, and we have used it successfully with SWRL ontologies containing up to 100 axioms, rules and facts. However, while it is useful to have a prototype that can be used for illustrative and test purposes, the effectiveness of such a naive approach must be open to question with larger SWRL ontologies.

In [55] it was shown that, when using the same translation approach to reason with OWL DL ontologies, performance could be greatly improved by using a so-called "relevant only" translation. The key idea is that when ontologies are translated, Vampire receives *all* of the axioms that occur in the ontology, whereas usually only a small fraction of them are actually relevant to a given subsumption or inconsistency problem. Vampire is not optimised to deal efficiently with large numbers of irrelevant axioms, and so it does not perform well under these circumstances.

An obvious way to correct this situation is to remove all irrelevant information from the FO task given to Vampire. An axiom is said to be *irrelevant* to a consistency test of C if it can easily be shown (i.e., via a syntactic analysis) that removing it from the ontology would not affect the interpretation of C; other axioms are called *relevant*. Note that not every "relevant axiom" really *will* affect the computation of the consistency of C, but we cannot (easily) rule out the possibility that it *may* affect the computation. An FO-translation is called *relevant-only* if it contains only FO-translations of axioms relevant (in the above sense) to the given satisfiability test.

The definition of relevance given in [55] can be extended to SWRL by treating rules in the same way as general concept inclusion axioms (GCIs). A concept or role expression *depends* on every concept or role that occurs in it, and a concept or role C depends on a concept or role D if D occurs in the definition of C. In addition, a concept C depends on every GCI and rule in the ontology.[19] *Relevance* is the transitive closure of depends. The process of selecting information relevant to a concept expression E looks very much the same as *unfolding* (see [1]), and assumes that the KB is separated into a set of unfoldable axioms and a set of GCIs [25] and rules. Every concept name CN and role name RN appearing in E is relevant to E. The process is then repeated recursively for unfoldable axioms with CN on the left hand side (whether inclusion or equality axioms). Also, if role R is relevant to E, then so are all roles R' s.t. $R \sqsubseteq R'$, along with their inverses (if the target DL allows inverse roles). An algorithm for computing relevant information is quite straightforward and is described in detail in [54].

Computing relevance leads to a small overhead when translating a SWRL ontology into FOL, but it should greatly increase the performance of the FO prover. Preliminary experiments with an extension of Hoolet to include an implementation of the relevant only translation suggest that this is indeed the case [29].

11 Discussion

In this paper we have presented SWRL, a proposed extension to OWL to include a simple form of Horn-style rules. We have provided formal syntax and semantics for SWRL, shown how OWL's XML and RDF syntax can be extended to deal with SWRL,

[19] It should be possible to treat (some) rules as unfoldable axioms, add thus eliminate the need to include all rules in the relevant only translation, but this is still the subject of ongoing work.

illustrated the features of SWRL with several examples, and discussed how reasoning support for SWRL might be provided.

The main strengths of the proposal are its simplicity and its tight integration with the existing OWL language. As we have seen, SWRL extends OWL with the most basic kind of Horn rule (sweetened with a little "syntactic sugar"): predicates are limited to being OWL classes and properties (and so have a maximum arity of 2), there are no disjunctions or negations (of atoms), no built in predicates (such as arithmetic predicates), and no nonmonotonic features such as negation as failure or defaults. Moreover, rules are given a standard first order semantics. This facilitates the tight integration with OWL, with SWRL being defined as a syntactic and semantic extension of OWL DL.

While we believe that SWRL defines a natural and useful level in the hierarchy of Semantic Web languages, it is clear that some applications would benefit from further extensions in expressive power. In particular, the ability to express arithmetic relationships between data values is important in many applications (e.g., to assert that persons whose income at least equals their expenditure are happy, while those whose expenditure exceeds their income are unhappy). It is not clear, however, if this would best be achieved by extending SWRL to include rules with built in arithmetic predicates, or by extending OWL Datatypes to include nary predicates [35].

Finally, we have shown how a first order theorem prover can be used to provide reasoning services for SWRL, and how some simple optimisations can be used to improve performance. Our results were sufficiently encouraging to suggest that, with further tuning and optimisation, such a strategy would be useful in (some) realistic applications. Future work will include such tuning and optimisation, as well as empirical investigations to determine the practical value of the resulting system.

Acknowledgements

This document has benefited from extensive discussion in the Joint US/EU ad hoc Agent Markup Language Committee. Parts of Section 9, in particular, were the result of feedback from and discussions with Benjamin Grosof and Mike Dean.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2002.

[2] F. Baader and U. Sattler. Number restrictions on complex roles in description logics: A preliminary report. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 328–338, 1996.

[3] L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.

[4] R. Berger. The undecidability of the dominoe problem. *Mem. Amer. Math. Soc.*, 66:1–72, 1966.

[5] T. Berners-Lee. Semantic web roadmap, 1998. Available at `http://www.w3.org/DesignIssues/Semantic`.

[6] P. V. Biron and A. Malhotra. XML schema part 2: Datatypes. W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-2/.

[7] A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[8] A. Borgida and P. F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *J. of Artificial Intelligence Research*, 1:277–308, 1994.

[9] The CASC web page. http://www.cs.miami.edu/~tptp/CASC/.

[10] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. W3C Recommendation, 10 February 2004. http://www.w3.org/TR/owl-ref/.

[11] H. Ganzinger and H. de Nivelle. A superposition decision procedure for the guarded fragment with equality. In *Proc. IEEE Conference on Logic in Computer Science (LICS)*, pages 295–304, 1999.

[12] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format Version 3.0 Reference Manual. Technical Report Logic Group Report Logic-92-1, Stanford University, 2001. http://logic.stanford.edu/kif/Hypertext/kif-manual.html.

[13] E. Gradel, M. Otto, and E. Rosen. Two-variable logic with counting is decidable. In *Proceedings of LICS 1997*, pages 306–317, 1997.

[14] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003.

[15] P. Hayes. RDF model theory. W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/rdf-mt/.

[16] L. Hollink, G. Schreiber, J. Wielemaker, and B. Wielinga. Semantic annotation of image collections. In *Workshop on Knowledge Markup and Semantic Annotation, KCAP'03*, 2003. Available at http://www.cs.vu.nl/~guus/papers/Hollink03c.pdf.

[17] M. Hori, J. Euzenat, and P. F. Patel-Schneider. OWL web ontology language XML presentation syntax. W3C Note, 11 June 2003. Available at http://www.w3.org/TR/owl-xmlsyntax/.

[18] I. Horrocks and P. F. Patel-Schneider. The generation of DAML+OIL. In *Proc. of the 2001 Description Logic Workshop (DL 2001)*, pages 30–35. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-49/, 2001.

[19] I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 17–29. Springer, 2003.

[20] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[21] I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204, 2001.

[22] I. Horrocks and U. Sattler. The effect of adding complex role inclusion axioms in description logics. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 343–348. Morgan Kaufmann, Los Altos, 2003.

[23] I. Horrocks and U. Sattler. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. *Artificial Intelligence*, 160(1–2):79–104, Dec. 2004.

[24] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.

[25] I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 285–296, 2000.

[26] U. Hustadt and R. A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 67–71. Springer, 2000.

[27] G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-concepts/`.

[28] A. Y. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.

[29] K. Li. Optimising first order reasoning over owl ontologies. Master's thesis, University of Manchester, 2004.

[30] D. V. McDermott and D. Dou. Representing disjunction and quantifiers in rdf. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2002)*, volume 2342 of *Lecture Notes in Computer Science*, pages 250–263. Springer, 2002.

[31] M. Mortimer. On languages with two variables. *Zeitschr. für math. Logik und Grundlagen der Math.*, 21:135–140, 1975.

[32] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. SETHEO and e-SETHEO - the CADE-13 systems. *Journal of Automated Reasoning*, 18(2):237–246, 1997.

[33] R. Nieuwenhuis and A. Rubio. Paramodulation-Based Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science, 2001.

[34] L. Padgham and P. Lambrix. A framework for part-of hierarchies in terminological logics. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 485–496, 1994.

[35] J. Pan and I. Horrocks. Web ontology reasoning with datatype groups. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 47–63. Springer, 2003.

[36] M. Paramasivam and D. A. Plaisted. Automated deduction techniques for classification in description logic systems. *J. of Automated Reasoning*, 20(3):337–364, 1998.

[37] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL web ontology language semantics and abstract syntax. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-semantics/`.

[38] P. F. Patel-Schneider, D. L. McGuiness, R. J. Brachman, L. A. Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bull.*, 2(3):108–113, 1991.

[39] A. Rector. Analysis of propagation along transitive roles: Formalisation of the galen experience with medical ontologies. In *Proc. of DL 2002*. CEUR (`http://ceur-ws.org/`), 2002.

[40] A. Rector, S. Bechhofer, C. A. Goble, I. Horrocks, W. A. Nowlan, and W. D. Solomon. The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9:139–171, 1997.

[41] A. Rector and I. Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proc. of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*, 1997.

[42] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[43] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[44] U. Sattler. Description logics for the representation of aggregated objects. In *Proc. of ECAI 2000*. IOS Press, 2000.

[45] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471, 1991.

[46] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'89)*, pages 421–431. Morgan Kaufmann, Los Altos, 1989.

[47] M. K. Smith, C. Welty, and D. L. McGuinness. OWL web ontology language guide. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-guide/`.

[48] K. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with snomed-rt. *J. of the Amer. Med. Informatics Ass.*, 2000. Fall Symposium Special Issue.

[49] G. Sutcliffe and C. Suttner. The TPTP Problem Library. TPTP v. 2.4.1. Technical report, University of Miami, 2001.

[50] T. Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.

[51] The DAML Services Coalition. Daml-s: Semantic markup for web services, May 2003. Available at `http://www.daml.org/services/daml-s/0.9/daml-s.html`.

[52] Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, 6 October 2000. Available at `http://www.w3.org/TR/REC-xml`.

[53] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.

[54] D. Tsarkov and I. Horrocks. DL reasoner vs. first-order prover. In *Proc. of the 2003 Description Logic Workshop (DL 2003)*, volume 81 of *CEUR (`http://ceur-ws.org/`)*, pages 152–159, 2003.

[55] D. Tsarkov, A. Riazanov, S. Bechhofer, and I. Horrocks. Using Vampire to reason with OWL. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proc. of the 2004 International Semantic Web Conference (ISWC 2004)*, number 3298 in Lecture Notes in Computer Science, pages 471–485. Springer, 2004.

Appendix B:

"Introducing Customised Datatypes and Datatype Predicates into OWL" by Jeff Z. Pan and Ian Horrocks, 2005 OWL: Experiences and Directions Workshop, Galway, Ireland, November 2005. Enclosed with the full electronic version of this report as the file owled2005-datatypes.pdf.

Introducing Customised Datatypes and Datatype Predicates into OWL(*)

Jeff Z. Pan and Ian Horrocks

School of Computer Science, University of Manchester, UK

Abstract. Although OWL is rather expressive, it has a very serious limitation on datatypes; i.e., it does not support customised datatypes. It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome, and the W3C Semantic Web Best Practices and Deployment Working Group has set up a task force to address this issue. This paper provides a solution for this issue by presenting two decidable datatype extensions of OWL DL, namely OWL-Eu and OWL-E. OWL-Eu provides a minimal extension of OWL DL to support customised datatypes, while OWL-E extends OWL DL with both customised datatypes and customised datatype predicates.

1 Introduction

The OWL Web Ontology Language [1] is a W3C recommendation for expressing ontologies in the Semantic Web. Datatype support [7, 8] is one of the key features that OWL is expected to provide, and has prompted extensive discussions in the RDF-Logic mailing list [10] and in the Semantic Web Best Practices mailing list [12]. Although OWL adds considerable expressive power to the Semantic Web, the OWL datatype formalism (or simply *OWL datatyping*) is much too weak for many applications; in particular, OWL datatyping does not provide a general framework for customised datatypes, such as XML Schema derived datatypes.

It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome [11], as it is often necessary to enable users to define their own datatypes and datatype predicates for their ontologies and applications. One of the most well known type systems is W3C XML Schema Part 2 [2], which defines facilities to allow users to define customised datatypes, such as those defined by imposing some restrictions in the value spaces of existing datatypes.

Example 1. Customised datatypes are useful in capturing the intended meaning of some vocabulary in ontologies. For example, users might want to use the customised datatype 'atLeast18' in the following definition of the class 'Adult':

```
Class(Adult complete Person
            restriction(age allValuesFrom(atLeast18))),
```

which says that an Adult is a Person whose age is at least 18. The datatype constraint

(*) This work is partially supported by the FP6 Network of Excellence EU project Knowledge Web (IST-2004-507842).

'at least 18' can be defined as an XML Schema user-defined datatype

```
<simpleType name = "atLeast18">
  <restriction base = "xsd:integer">
    <minInclusive value = "18"/>
  </restriction>
</simpleType>
```

in which the facet 'minInclusive' is used to restrict the value space of `atLeast18` (a customised datatype) to be a subset of the value space of `integer` (an XML Schema built-in datatype).

User-defined datatypes (like the above one) cannot, however, be used in the OWL datatyping, which (only) provides the use of some *built-in* XML Schema datatypes and enumerated datatypes, which are defined by explicitly specifying their instances. The OWL datatyping does not support XML Schema customised datatypes for the following two reasons: (i) XML Schema does not provide a standard way to access a user-defined datatype. (ii) OWL DL does not provide a mechanism to guarantee the computability of the kinds of customised datatypes it supports.

This paper provides a solution for this issue by presenting two decidable datatype extensions of OWL DL, namely OWL-Eu and OWL-E. OWL-Eu provides a minimal extension of OWL DL to support customised datatypes, while OWL-E extends OWL DL with both customised datatypes and customised datatype predicates. The rest of the paper is organised as follows: Section 2 further discusses the motivations of introducing customised datatypes and datatype predicates. Section 3 extends the OWL datatyping to unary datatype groups, which enables the use of customised datatypes. Section 4 and 5 present the OWL-Eu and the OWL-E languages, respectively; the latter one is based on datatype groups, which are general forms of unary datatype groups. Section 6 concludes the paper and suggests some future work.

2 Motivations

Allowing users to define their own vocabulary is one of the most useful features that ontologies can provide over other approaches, such as the Dublin Core, of providing semantics in the Semantic Web. In the Dublin Core standard, the meaning of the set of 15 information properties are described in English text. The main drawback of the Dublin Core is its inflexibility; it is impossible to 'predefine' information properties for all sorts of applications.

Ontologies, however, are more flexible in that users can define their own vocabulary based on existing vocabularies. In ontology languages, a set of class constructors are usually provided so that users can build class expressions based on, for example, existing class names. The intended meaning of the vocabulary, therefore, can be captured by the axioms in the ontologies. Let us revisit Example 1 and consider the intended meaning of the Adult class. According to its definition, an Adult is a Person who is at least 18 years old. As a result, programs can also understand the meaning of customised vocabulary, with the help of ontologies.

Although OWL DL provides a set of expressive class constructors to build customised classes, it does not provide enough expressive power to support, for example,

XML Schema customised datatypes. In order to capture the intended meaning of Adult, Example 1 has already shown the necessity of customised datatypes. In what follows, we give some more examples to illustrate the usefulness of customised datatypes and datatype predicates in various SW and ontology applications.

Example 2. **Semantic Web Service: Matchmaking**

Matchmaking is a process that takes a service requirement and a group of service advertisements as input, and returns all the advertisements that may potentially satisfy the requirement. In a computer sales ontology, a service requirement may ask for a PC with memory size greater than 512Mb, unit price less than 700 pounds and delivery date earlier than 15/03/2004.

Here 'greater than 512', 'less than 700' and 'earlier than 15/03/2004' are customised datatypes of base datatypes integer, integer and date, respectively.

Example 3. **Electronic Commerce: A 'No Shipping Fee' Rule**

Electronic shops may need to classify items according to their sizes, and to reason that an item for which the sum of height, length and width is no greater than 15cm belongs to a class in their ontology, called 'small-items'. Then they can have a rule saying that for 'small-items' no shipping costs are charged. Accordingly, the billing system will charge no shipping fees for all the instances of the 'small-items' class.

Here 'greater than 15' is a customised datatype, 'sum' is a datatype predicate, while 'sum no greater than 15' is a customised datatype predicate.

3 Unary Datatype Groups

The OWL datatyping is defined based on the notion of datatype maps [9]. A datatype map is a partial mapping from supported datatype URIrefs to datatypes. In this section, we introduce unary datatype groups, which extend the OWL datatyping with a hierarchy of supported datatypes.

Definition 1 A *unary datatype group* $\mathcal{G}$ is a triple ($\mathbf{M}_d$,$\mathbf{B}$,dom), where $\mathbf{M}_d$ is the *datatype map* of $\mathcal{G}$, $\mathbf{B}$ is the set of *primitive base datatype* URI references in $\mathcal{G}$ and dom is the *declared domain function*. We call $\mathbf{S}$ the set of supported datatype URI references of $\mathcal{G}$, i.e., for each $u \in \mathbf{S}$, $\mathbf{M}_d(u)$ is defined; we require $\mathbf{B} \subseteq \mathbf{S}$. We assume that there exist unary datatype URI reference $\mathrm{rdfs{:}Literal}$, $\mathrm{owlx{:}DatatypeBottom} \notin \mathbf{S}$. The declared domain function dom has the following properties: for each $u \in \mathbf{S}$, if $u \in \mathbf{B}$, $\mathsf{dom}(u) = u$; otherwise, $\mathsf{dom}(u) = v$, where $v \in \mathbf{B}$. $\diamond$

Definition 1 ensures that all the primitive base datatype URIrefs of $\mathcal{G}$ are supported ($\mathbf{B} \subseteq \mathbf{S}$) and that each supported datatype URIref relates to a primitive base datatype URIref through the declared domain function dom.

Example 4. $\mathcal{G}_1 = (\mathbf{M}_{d1}, \mathbf{B}_1, \mathsf{dom}_1)$ is a unary datatype group, where

- $\mathbf{M}_{d1} = \{\mathrm{xsd{:}integer} \mapsto \texttt{integer}, \mathrm{xsd{:}string} \mapsto \texttt{string}, \mathrm{xsd{:}nonNegativeInteger} \mapsto \geq_0, \mathrm{xsdx{:}integerLessThanN} \mapsto <_{\mathrm{N}}\}$,
- $\mathbf{B}_1 = \{\mathrm{xsd{:}string}, \mathrm{xsd{:}integer}\}$, and

– $\mathsf{dom}_1 = \{\text{xsd:integer} \mapsto \text{xsd:integer}, \text{xsd:string} \mapsto \text{xsd:string}, \text{xsd:nonNegativeInteger} \mapsto \text{xsd:integer}, \text{xsdx:integerLessThanN} \mapsto \text{xsd:integer}\}$.

According to $\mathbf{M}_{d1}$, we have $\mathbf{S}_1 = \{\text{xsd:integer}, \text{xsd:string}, \text{xsd:nonNegativeInteger}, \text{xsdx:integerLessThanN}\}$, hence $\mathbf{B}_1 \subseteq \mathbf{S}_1$. ♢

Based on a unary datatype group, we can provide a formalism (called datatype expressions) for constructing customised datatypes using supported datatypes.

Definition 2 Let $\mathcal{G}$ be a unary datatype group. The set of *$\mathcal{G}$-unary datatype expressions* in abstract syntax (corresponding DL syntax can be found in Table 5 on page 8), abbreviated $\mathbf{Dexp}(\mathcal{G})$, is inductively defined as follows:

1. *atomic expressions* $u \in \mathbf{Dexp}(\mathcal{G})$, for a datatype URIref u;
2. *relativised negated expressions* $\mathtt{not}(u) \in \mathbf{Dexp}(\mathcal{G})$, for a datatype URIref u;
3. *enumerated expressions* $\mathtt{oneOf}(l_1, \ldots, l_n) \in \mathbf{Dexp}(\mathcal{G})$, for literals $l_1, \ldots, l_n$;
4. *conjunctive expressions* $\mathtt{and}(E_1, ..., E_n) \in \mathbf{Dexp}(\mathcal{G})$, for unary datatype expressions $E_1, ..., E_n \in \mathbf{Dexp}(\mathcal{G})$;
5. *disjunctive expressions* $\mathtt{or}(E_1, ..., E_n) \in \mathbf{Dexp}(\mathcal{G})$, for unary datatype expressions $E_1, ..., E_n \in \mathbf{Dexp}(\mathcal{G})$. ⋄

Example 5. $\mathcal{G}$-unary datatype expressions can be used to represent XML Schema non-list simple types. Given the unary datatype group $\mathcal{G}_1$ presented in Example 4 (page 3),

– the following XML Schema derived union simple type

```
<simpleType name = "cameraPrice">
  <union>
    <simpleType>
      <restriction base = "xsd:nonNegativeInteger">
        <maxExclusive value = "100000"/>
      </restriction>
    </simpleType>
    <simpleType>
      <restriction base = "xsd:string">
        <enumeration value = "low"/>
        <enumeration value = "medium"/>
        <enumeration value = "expensive"/>
      </restriction>
    </simpleType>
  </union>
<simpleType>
```

can be represented by the following disjunctive expression

or(
  and(xsd:nonNegativeInteger, xsdx:integerLessThan100000)
  oneOf("low"^^xsd:string,"medium"^^xsd:string, "expensive"^^xsd:string)
).

Note that "low"^^xsd:string is a typed literal, which represents a value of the xsd:string datatype. "low", instead, is a plain literal, where no datatype information is provided. ♢

We now define the interpretation of a unary datatype group.

Abstract Syntax	DL Syntax	Semantics
a datatype URIref u	u	$u^{\mathbf{D}}$
$\mathtt{oneOf}(l_1, \ldots, l_n)$	$\{l_1, \ldots, l_n\}$	$\{l_1^{\mathbf{D}}\} \cup \ldots \cup \{l_n^{\mathbf{D}}\}$
$\mathtt{not}$(u)	$\overline{u}$	$(\mathsf{dom}(u))^{\mathbf{D}} \setminus u^{\mathbf{D}}$ if $u \in \mathbf{S} \setminus \mathbf{B}$ $\Delta_{\mathbf{D}} \setminus u^{\mathbf{D}}$ otherwise
$\mathtt{and}(E_1, \ldots, E_n)$	$E_1 \wedge \ldots \wedge E_n$	$E_1^{\mathbf{D}} \cap \ldots \cap E_n^{\mathbf{D}}$
$\mathtt{or}(P, Q)$	$E_1 \vee \ldots \vee E_n$	$E_1^{\mathbf{D}} \cup \ldots \cup E_n^{\mathbf{D}}$

Table 1. Syntax and semantics of datatype expressions (OWL-Eu data ranges)

Definition 3 A *datatype interpretation* $\mathcal{I}_{\mathbf{D}}$ of a unary datatype group $\mathcal{G} = (\mathbf{M}_d, \mathbf{B}, \mathsf{dom})$ is a pair $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$, where $\Delta_{\mathbf{D}}$ (the datatype domain) is a non-empty set and $\cdot^{\mathbf{D}}$ is a datatype interpretation function, which has to satisfy the following conditions:

1. $(\mathrm{rdfs{:}Literal})^{\mathbf{D}} = \Delta_{\mathbf{D}}$ and $(\mathrm{owlx{:}DatatypeBottom})^{\mathbf{D}} = \emptyset$;
2. for each plain literal l, $l^{\mathbf{D}} = l \in \mathbf{PL}$ and $\mathbf{PL} \subseteq \Delta_{\mathbf{D}}$ ($\mathbf{PL}$ is the value space for plain literals);
3. for any two primitive base datatype URIrefs $u_1, u_2 \in \mathbf{B}$: $u_1^{\mathbf{D}} \cap u_2^{\mathbf{D}} = \emptyset$;
4. for each supported datatype URIref $u \in \mathbf{S}$, where $d = \mathbf{M}_d(u)$:
   (a) $u^{\mathbf{D}} = V(d) \subseteq \Delta_{\mathbf{D}}$, $L(u) \subseteq L(\mathsf{dom}(u))$ and $L2V(u) \subseteq L2V(\mathsf{dom}(u))$;
   (b) if $s \in L(d)$, then $(\text{“s”}\hat{}\hat{}u)^{\mathbf{D}} = L2V(d)(s)$; otherwise, $(\text{“s”}\hat{}\hat{}u)^{\mathbf{D}}$ is not defined;
5. $\forall u \notin \mathbf{S}$, $u^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$, and $\text{“v”}\hat{}\hat{}u \in u^{\mathbf{D}}$.

Moreover, we extend $\cdot^{\mathbf{D}}$ to $\mathcal{G}$ unary datatype expression as shown in Table 5 (page 8). Let E be a $\mathcal{G}$ unary datatype expression, the negation of E is of the form $\neg E$, which is interpreted as $\Delta_{\mathbf{D}} \setminus E^{\mathbf{D}}$. ◇

Next, we introduce the kind of basic reasoning mechanisms required for a unary datatype group.

Definition 4 Let $\mathbf{V}$ be a set of variables, $\mathcal{G} = (\mathbf{M}_d, \mathbf{B}, \mathsf{dom})$ a unary datatype group and $u \in \mathbf{B}$ a primitive base datatype URIref. A datatype conjunction of u is of the form

$$\mathscr{C} = \bigwedge_{j=1}^{k} u_j(v_j) \wedge \bigwedge_{i=1}^{l} \neq_i (v_1^{(i)}, v_2^{(i)}), \tag{1}$$

where the v_j are variables from $\mathbf{V}$, $v_1^{(i)}, v_2^{(i)}$ are variables in $\bigwedge_{j=1}^{k} u_j(v_j)$, u_j are datatype URI references from $\mathbf{S}$ such that $\mathsf{dom}(u_j) = u$, and $\neq_i$ are the inequality predicates for primitive base datatypes $\mathbf{M}_d(\mathsf{dom}(u_i))$ where u_i appear in $\bigwedge_{j=1}^{k} u_j(v_j)$.

A datatype conjunction $\mathscr{C}$ is called *satisfiable* iff there exists an interpretation $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$ of $\mathcal{G}$ and a function δ mapping the variables in $\mathscr{C}$ to data values in $\Delta_{\mathbf{D}}$ s.t. $\delta(v_j) \in u_j^{\mathbf{D}}$ (for all $1 \leq j \leq k$) and $\{\delta(v_1^{(i)}), \delta(v_2^{(i)})\} \subseteq u_i^{\mathbf{D}}$ and $\delta(v_1^{(i)}) \neq \delta(v_2^{(i)})$ (for all $1 \leq i \leq l$). Such a function δ is called a *solution* for $\mathscr{C}$ w.r.t. $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$. ◇

We end this section by elaborating the conditions that computable unary datatype groups require.

Definition 5 A unary datatype group $\mathcal{G}$ is *conforming* iff

1. for any $u \in \mathbf{S} \setminus \mathbf{B}$: there exists $u' \in \mathbf{S} \setminus \mathbf{B}$ such that $u'^{\mathbf{D}} = \overline{u}^{\mathbf{D}}$, and
2. for each primitive base datatype in $\mathcal{G}$, the satisfiability problems for finite datatype conjunctions of the form (1) is decidable. $\diamond$

4 OWL-Eu

In this section, we present a small extension of OWL DL, i.e., OWL-Eu. The underpinning DL of OWL-Eu is $\mathcal{SHOIN}(\mathcal{G}_1)$, i.e., the $\mathcal{SHOIN}$ DL combined with a unary datatype group $\mathcal{G}$ (1 for unary). Specifically, OWL-Eu (only) extends OWL data range (i.e., enumerated datatypes as well as some built-in XML Schema datatypes) to OWL-Eu data ranges defined as follows.

Definition 6 An *OWL-Eu data range* is a $\mathcal{G}$ unary datatype expression. Abstract (as well as DL) syntax and model-theoretic semantics of OWL-Eu data ranges are presented in Table 5 (page 8). $\diamond$

The consequence of the extension is that customised datatypes, represented by OWL-Eu data ranges, can be used in datatype exists restrictions ($\exists T.u$) and datatype value restrictions ($\forall T.u$), where T is a datatype property and u is an OWL-Eu data range. Hence, this extension of OWL DL is as large as is necessary to support customised datatypes.

Example 6. PCs with memory size greater than or equal to 512 Mb and with price cheaper than 700 pounds can be represented in the following OWL-Eu concept description in DL syntax (cf. Table 5 on page 8):

$$\mathsf{PC} \sqcap \exists memorySizeInMb.\overline{<_{512}} \sqcap \exists priceInPound. <_{700},$$

where $\overline{<_{512}}$ is a relativised negated expression and $<_{700}$ is a supported datatype in $\mathcal{G}_1$. $\diamondsuit$

It turns out that OWL-Eu (i.e., the $\mathcal{SHOIN}(\mathcal{G}_1)$ DL) is decidable.

Theorem 1. *The $\mathcal{SHOIN}(\mathcal{G}_1)$-concept satisfiability problem w.r.t. a knowledge base is decidable if the combined unary datatype group is conforming.*

Proof: (Sketch) We will show the decidability of $\mathcal{SHOIN}(\mathcal{G}_1)$-concept satisfiability w.r.t. TBoxes and RBoxes by reducing it to the $\mathcal{SHOIN}$-concept satisfiability w.r.t. TBoxes and RBoxes. The basic idea behind the reduction is that we can replace each datatype group-based concept C in $\mathcal{T}$ with a new atomic primitive concept A_C in $\mathcal{T}'$. We then compute the satisfiability problem for all possible conjunctions of datatype group-based concepts (and their negations) in $\mathcal{T}$ (of which there are only a finite number), and in case a conjunction $C_1 \sqcap \ldots \sqcap C_n$ is unsatisfiable, we add an axiom $A_{C_1} \sqcap \ldots \sqcap A_{C_n} \sqsubseteq \bot$ to $\mathcal{T}'$. For example, unary datatype group-based concepts $\exists T.\ >_1$ and $\forall T.\ \leq_0$ occurring in $\mathcal{T}$ would be replaced with $A_{\exists T.>_1}$ and $A_{\forall T.\leq_0}$ in $\mathcal{T}'$, and $A_{\exists T.>_1} \sqcap A_{\forall T.\leq_0} \sqsubseteq \bot$ would be added to $\mathcal{T}'$ because $\exists T.\ >_1 \sqcap \forall T.\ \leq_0$ is *unsatisfiable* (i.e., there is no solution for the predicate conjunction $>_1 (v) \wedge \leq_0 (v)$).

5 OWL-E: A Step Further

In this section, we present a further extension of OWL-Eu, called OWL-E, which supports not only customised datatypes, but also customised datatype predicates.

A *datatype predicate* (or simply *predicate*) p is characterised by an arity $a(p)$, or a minimum arity $a_{min}(p)$ if p can have multiple arities, and a predicate extension (or simply *extension*) $E(p)$. The notion of predicate maps can be defined in an obvious way. For example, $=^{int}$ is a (binary) predicate with arity $a(=^{int}) = 2$ and extension $E(=^{int}) = \{\langle i_1, i_2\rangle \in V(\texttt{integer})^2 \mid i_1 = i_2\}$, where $V(\texttt{integer})$ is the value space for the datatype `integer`.

Now we can generalise unary datatype groups by the definition of datatype groups. In fact, datatypes and datatype predicates can be unified in datatype groups. Roughly speaking, a datatype group is a group of built-in predicate URIrefs 'wrapped' around a set of primitive datatype URIrefs. A *datatype group* $\mathcal{G}$ is a tuple ($\mathbf{M}_p$,$\mathbf{B}$,dom), where $\mathbf{M}_p$ is the *predicate map* of $\mathcal{G}$, $\mathbf{B}$ is the set of *primitive datatype* URI references in $\mathcal{G}$ and dom is the *declared domain function*. We call $\mathbf{S}$ the set of built-in predicate URI references of $\mathcal{G}$, i.e., for each $u \in \mathbf{S}$, $\mathbf{M}_p(u)$ is defined; we require $\mathbf{B} \subseteq \mathbf{S}$. The declared domain function dom has the following properties: for each $u \in \mathbf{S}$,

$$\mathsf{dom}(u) = \begin{cases} u & \text{if } u \in \mathbf{B}, \\ (v_1, \ldots, v_n), \text{ where } v_1, \ldots, v_n \in \mathbf{B} & \text{if } u \in \mathbf{S} \setminus \mathbf{B} \text{ and} \\ & a(\mathbf{M}_p(u)) = n, \\ \{\underbrace{(v, \ldots, v)}_{i \text{ times}} \mid i \geq n\}, \text{ where } v \in \mathbf{B} & \text{if } u \in \mathbf{S} \setminus \mathbf{B} \text{ and} \\ & a_{min}(\mathbf{M}_p(u)) = n. \end{cases}$$

Example 7. $\mathcal{G}_2 = (\mathbf{M}_{p_2}, \mathbf{B}_2, \mathsf{dom}_2)$ is a datatype group, where

- $\mathbf{M}_{p_2} = \{\mathrm{xsd{:}integer} \mapsto \texttt{integer}, \mathrm{xsd{:}string} \mapsto \texttt{string}, \mathrm{xsd{:}integerGreaterThanOrEqualToN} \mapsto \geq_{\mathrm{N}}, \mathrm{xsdx{:}integerLessThanN} \mapsto <_{\mathrm{N}}, \mathrm{xsdx{:}integerEquality} \mapsto =^{\mathtt{int}}\}$,
- $\mathbf{B}_2 = \{\mathrm{xsd{:}string}, \mathrm{xsd{:}integer}\}$, and
- $\mathsf{dom}_2 = \{\mathrm{xsd{:}integer} \mapsto \mathrm{xsd{:}integer}, \mathrm{xsd{:}string} \mapsto \mathrm{xsd{:}string}, \mathrm{xsd{:}integerGreaterThanOrEqualToN} \mapsto \mathrm{xsd{:}integer}, \mathrm{xsdx{:}integerLessThanN} \mapsto \mathrm{xsd{:}integer}, \mathrm{xsdx{:}integerEquality} \mapsto (\mathrm{xsd{:}integer}, \mathrm{xsd{:}integer})\}$.

According to $\mathbf{M}_{p_2}$, we have $\mathbf{S}_2 = \{\mathrm{xsd{:}integer}, \mathrm{xsd{:}string}, \mathrm{xsd{:}nonNegativeInteger}, \mathrm{xsdx{:}integerLessThanN}, \mathrm{xsdx{:}integerEquality}\}$, hence $\mathbf{B}_2 \subseteq \mathbf{S}_2$. ♢

Furthermore, based on datatype groups, we can extend unary datatype expressions to general (n-ary) datatype expressions. While enumerated expressions remain the same, relativised negated, conjunctive and disjunctive unary datatype expressions can be easily extended to the n-nary case. There is a new kind of datatype expression called *domain expression*: $\texttt{domain}(u_1, \ldots, u_n)$, where u_i is either $\mathrm{rdfs{:}Literal}$ or supported unary datatype predicate URIrefs, or their relativised negations. For example, the customised predicate 'sumNoGreaterThanOrEqualTo15', with extension $E(sumNoGreaterThanOrEqualTo15) = \{\langle i_0, i_1, i_2, i_3\rangle \in V(\texttt{integer})^4 \mid i_0 =$

Abstract Syntax	DL Syntax	Semantics
$\mathtt{not}(u)$	$\overline{u}$	$\Delta_{\mathbf{D}} \setminus u^{\mathbf{D}}$ if $u \in \mathbf{B}$ $(\mathsf{dom}(u))^{\mathbf{D}} \setminus u^{\mathbf{D}}$ if $u \in \mathbf{S} \setminus \mathbf{B}$ $\bigcup_{n\geq 1}(\Delta_{\mathbf{D}})^n \setminus u^{\mathbf{D}}$ otherwise
$\mathtt{domain}(u_1, \ldots, u_n)$	$[u_1, \ldots, u_n]$	$u_1^{\mathbf{D}} \times \ldots \times u_n^{\mathbf{D}}$

Table 2. Syntax and semantics of (new) datatype expressions

$i_1 + i_2 + i_3$ and $\neg(i_0 \geq 15)\}$ and arity $a(sumNoGreaterThanOrEqualTo15) = 4$, can be represented by

$$\begin{array}{c}\mathrm{xsdx{:}integerAddition} \wedge \\ [\ \overline{\mathrm{xsdx{:}integerGreaterThanOrEqualTo15}}, \mathrm{xsd{:}integer}, \mathrm{xsd{:}integer}, \mathrm{xsd{:}integer}\ ],\end{array}$$

which is a conjunctive expression, where the first conjunct is a predicate URIref (that represents $+^{int}$) and the second conjunct is a domain expression.

We can extend the datatype interpretation $\mathcal{I}_{\mathbf{D}}$ presented in Definition 3 to give semantics to datatype groups. For each $u \in \mathbf{S}$, $u^{\mathbf{D}} = E(\mathbf{M}_p(u)) \subseteq (\mathsf{dom}(u))^{\mathbf{D}}$, where $(\mathsf{dom}(u))^{\mathbf{D}}$ is defined as follows: if $\mathsf{dom}(u) = (d_1, \ldots, d_n)$ and $a(\mathbf{M}_p(u)) = n$, then $(\mathsf{dom}(u))^{\mathbf{D}} = d_1^{\mathbf{D}} \times \ldots \times d_n^{\mathbf{D}}$; if $\mathsf{dom}(u) = \{(\underbrace{d, \ldots, d}_{i \text{ times}}) \mid i \geq n\}$ and $a_{min}(\mathbf{M}_p(u)) = n$, then $(\mathsf{dom}(u))^{\mathbf{D}} = \bigcup_{i\geq n} (d^{\mathbf{D}})^i$. The abstract syntax, DL syntax and semantics of relativised negated and domain expressions are presented in Table 5.

The following definition summarises the conditions that computable datatype groups require.

Definition 7 (Conforming Datatype Group) A datatype group $\mathcal{G}$ is *conforming* iff

1. for any $u \in \mathbf{S} \setminus \mathbf{B}$ with $a(\mathbf{M}_p(u)) = n \geq 2$: $\mathsf{dom}(u) = (\underbrace{w, \ldots, w}_{n \text{ times}})$ for some $w \in \mathbf{B}$, and
2. for any $u \in \mathbf{S} \setminus \mathbf{B}$: there exist $u' \in \mathbf{S} \setminus \mathbf{B}$ such that $u'^{\mathbf{D}} = \overline{u}^{\mathbf{D}}$, and
3. the satisfiability problem for finite negation-free predicate conjunctions is decidable, and
4. for each primitive datatype URIref $u_i \in \mathbf{B}$, there exists $w_i \in \mathbf{S}$, s.t. $\mathbf{M}_p(w_i) = \neq_{u_i}$ where $\neq_{u_i}$ is the binary inequality predicate for $\mathbf{M}_p(u_i)$. $\diamond$

Finally, OWL-E extends OWL-Eu with the datatype group-related class constructors presented in Table 3.

Example 8. (OWL-E classes)

Assume that electronic-shops want to define small items as items of which the sum of height, length and width is no greater than or equal to 15cm. The SmallItem class can be represented by the following datatype group-based concept description:

$$\exists T_s, T_h, T_l, T_w.(+^{int} \wedge [\overline{\geq_{15}}, integer, integer, integer]),$$

New Element	DL Syntax	Semantics
expressive predicate exists restriction	$\exists T_1, \ldots, T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \exists t_1, \ldots, t_n.\langle x, t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \wedge \langle t_1, \ldots, t_n\rangle \in E^{\mathbf{D}}\}$
expressive predicate value restriction	$\forall T_1, \ldots, T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \forall t_1, \ldots, t_n.\langle x, t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \rightarrow \langle t_1, \ldots, t_n\rangle \in E^{\mathbf{D}}\}$
expressive predicate atleast restriction	$\geqslant m T_1, \ldots, T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{\langle t_1, \ldots, t_n\rangle \mid \langle x, t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \wedge \langle t_1, \ldots, t_n\rangle \in E^{\mathbf{D}}\} \geq m\}$
expressive predicate atmost restriction	$\leqslant m T_1, \ldots, T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{\langle t_1, \ldots, t_n\rangle \mid \langle x, t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \wedge \langle t_1, \ldots, t_n\rangle \in E^{\mathbf{D}}\} \leq m\}$

Table 3. New class constructors in OWL-E

where T_s, T_h, T_l, T_w are concrete roles representing "sum in cm", "hight in cm", "length in cm" and "width in cm", respectively, and $(+^{int} \wedge [\overline{\geq_{15}}, integer, integer, integer])$ is a conjunctive datatype expression representing the customised predicate "sum no larger than or equal to 15".[1] $\diamondsuit$

Like OWL-Eu, OWL-E (i.e., the $\mathcal{SHOIQ}(\mathcal{G})$ DL) is also a decidable extension of OWL-DL.

Theorem 2. *The $\mathcal{SHOIN}(\mathcal{G})$- and $\mathcal{SHOIQ}(\mathcal{G})$-concept satisfiability and subsumption problems w.r.t. TBoxes and RBoxes are decidable.*

According to Tobies [13, Lemma 5.3], if $\mathcal{L}$ is a DL that provides the nominal constructor, knowledge base satisfiability can be polynomially reduced to satisfiability of TBoxes and RBoxes. Hence, we obtain the following theorem.

Theorem 3. *The knowledge base satisfiability problems of $\mathcal{SHOIN}(\mathcal{G})$ and $\mathcal{SHOIQ}(\mathcal{G})$ are decidable.*

6 Conclusion

In this paper, we propose OWL-Eu and OWL-E, two decidable extensions of OWL DL that support customised datatypes and customised datatype predicates. OWL-Eu provides a general framework for integrating OWL DL with customised datatypes, such as XML Schema non-list simple types. OWL-E further extends OWL-Eu to support customised datatype predicates.

We have implemented a prototype extension of the FaCT [5] DL system, called FaCT-DG, to support TBox reasoning in both OWL-Eu and OWL-E (without nominals). As for future work, we are planning to extend the DIG1.1 interface [3] to support OWL-Eu, and to implement a Protégé [6] plug-in to support XML Schema non-list simple types, i.e. users should be able to define and/or import customised XML Schema non-list simple types based on a set of supported datatypes, and to exploit our prototype through the extended DIG interface. Furthermore, we plan to extend the FaCT++ DL reasoner [4] to support the full OWL-Eu and OWL-E ontology languages.

[1] To save space, we use predicates instead of predicate URIrefs here.

Bibliography

[1] Sean Bechhofer, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein eds. OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/, Feb 2004.

[2] Paul V. Biron and Ashok Malhotra. Extensible Markup Language (XML) Schema Part 2: Datatypes – W3C Recommendation 02 May 2001. Technical report, World Wide Web Consortium, 2001. http://www.w3.org/TR/xmlschema-2/.

[3] DIG. SourceForge DIG Interface Project. http://sourceforge.net/projects/dig/, 2004.

[4] FaCT++. `http://owl.man.ac.uk/factplusplus/`, 2003.

[5] Ian Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. of KR'98*, pages 636–647, 1998.

[6] Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *International Semantic Web Conference*, pages 229–243, 2004.

[7] Jeff Z. Pan and Ian Horrocks. Extending Datatype Support in Web Ontology Reasoning. In *Proc. of the 2002 Int. Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2002)*, Oct 2002.

[8] Jeff Z. Pan and Ian Horrocks. Web Ontology Reasoning with Datatype Groups. In *Proc. of the 2003 International Semantic Web Conference (ISWC2003)*, pages 47–63, 2003.

[9] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. Technical report, W3C, Feb. 2004. W3C Recommendation, URL `http://www.w3.org/TR/2004/REC-owl-semantics-20040210/`.

[10] RDF-Logic Mailing List. http://lists.w3.org/archives/public/www-rdf-logic/. W3C Mailing List, starts from 2001.

[11] Alan Rector. Re: [UNITS, OEP] FAQ : Constraints on data values range. Discussion in [12], Apr. 2004. http://lists.w3.org/Archives/Public/public-swbp-wg/2004Apr/0216.html.

[12] Semantic Web Best Practice and Development Working Group Mailing List. http://lists.w3.org/archives/public/public-swbp-wg/. W3C Mailing List, starts from 2004.

[13] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, Rheinisch-Westfälischen Technischen Hochschule Aachen, 2001. URL `http://lat.inf.tu-dresden.de/research/phd/Tobies-PhD-2001.pdf` .

Appendix C:

"The Irresistible SRIQ" by Ian Horrocks, Oliver Kutz, and Ulrike Sattler, 2005 OWL: Experiences and Directions Workshop, Galway, Ireland, November 2005. Enclosed with the full electronic version of this report as the file owled2005-sriq.pdf.

The Irresistible $\mathcal{SRIQ}$

Ian Horrocks, Oliver Kutz, and Ulrike Sattler

School of Computer Science, The University of Manchester,
Kilburn Building, Oxford Road, Manchester, M13 9PL, UK,
{Horrocks, Kutz, Sattler}@cs.man.ac.uk

Abstract. Motivated primarily by medical terminology applications, the prominent DL $\mathcal{SHIQ}$ has already been extended to a DL with complex role inclusion axioms of the form $R \circ S \dot{\sqsubseteq} R$ or $S \circ R \dot{\sqsubseteq} R$, called $\mathcal{RIQ}$, and the $\mathcal{SHIQ}$ tableau algorithm has been extended to handle such inclusions.
This paper further extends $\mathcal{RIQ}$ and its tableau algorithm with important expressive means that are frequently requested in ontology applications, namely with reflexive, symmetric, transitive, and irreflexive roles, disjoint roles, and the construct $\exists R.\mathsf{Self}$, allowing, for instance, the definition of concepts such as a "narcist". Furthermore, we extend the algorithm to cover Abox reasoning extended with negated role assertions. The resulting logic is called $\mathcal{SRIQ}$.

1 Introduction

We describe an extension, called $\mathcal{SRIQ}$, of the description logic (DL) $\mathcal{SHIN}$ (10) underlying OWL lite and OWL DL (7). We believe that $\mathcal{SRIQ}$ enjoys some useful properties. Firstly, $\mathcal{SRIQ}$ extends $\mathcal{SHIN}$ with numerous expressive means which have been asked for by users, and which, we believe, will make modeling using DLs easier and more intuitive. While the language of $\mathcal{SRIQ}$ is designed to be slightly redundant in the sense that some of the new expressive means can be simulated by others, the complete absence of those expressive means has proven quite harmful since developers of ontologies use work-arounds to compensate for this. As a consequence, ontologies become cluttered, complicated, and difficult to understand. In the worst case, the work-around only partially captures the intended semantics, thus leading to unintended or missing consequences, thereby destroying one of the main features of a logic-based formalism, namely its well-defined semantics and reasoning services. A well-known example are *qualified* number restrictions. Their absence in OWL lite and OWL DL has caused problems in the past (12), and has led to the development and use of questionable surrogates. Hence, $\mathcal{SRIQ}$ provides qualified number restrictions. Other, novel expressive means of $\mathcal{SRIQ}$ concern mostly roles and include:

– *disjoint roles.* E.g., the roles `sister` and `mother` could be declared as being disjoint. Most DLs can be said to be "lopsided" since they allow to express disjointness on concepts but not on roles, despite the fact that role disjointness is quite natural and can generate new subsumptions or inconsistencies in the presence of role hierarchies and number restrictions.

- *reflexive and irreflexive roles.* E.g., the role `knows` could be declared as being reflexive, and the role `sibling` could be declared as being irreflexive. In the presence of the new concept $\exists R.\mathsf{Self}$ described below, reflexive and irreflexive roles also become definable by Tbox assertions.
- *negated role assertions.* Most Abox formalisms only allow for positive role assertions (with few exceptions (1; 5)), whereas $\mathcal{SRIQ}$ also allows for statements such as $(\texttt{John}, \texttt{Mary}) : \neg\texttt{likes}$. In the presence of complex role inclusions, negated role assertions can be quite useful and, like disjoint roles, they overcome a certain "lopsidedness" of DLs.
- Since $\mathcal{SRIQ}$ extends $\mathcal{SHIQ}$, we can also express that a role is transitive or symmetric, and can use role inclusion axioms $R \sqsubseteq S$.
- Since $\mathcal{SRIQ}$ extends $\mathcal{RIQ}$ (8), we can use complex role inclusion axioms of the form $R \circ S \dot{\sqsubseteq} R$ and $S \circ R \dot{\sqsubseteq} R$. For example, w.r.t. the axiom $\texttt{owns} \circ \texttt{hasPart} \dot{\sqsubseteq} \texttt{owns}$, and the fact that each car contains an engine $\texttt{Car} \dot{\sqsubseteq} \exists\texttt{hasPart}.\texttt{Engine}$, an owner of a car is also an owner of an engine, i.e., the following subsumption is implied: $\exists\texttt{owns}.\texttt{Car} \sqsubseteq \exists\texttt{owns}.\texttt{Engine}$.
- Finally, $\mathcal{SRIQ}$ allows for concepts of the form $\exists R.\mathsf{Self}$ which can be used to express "local reflexivity" of a role R, e.g., to define the concept "narcist" using $\exists\texttt{likes}.\mathsf{Self}$.

Besides a Tbox and an Abox, $\mathcal{SRIQ}$ provides a so-called *Rbox* to gather all statements concerning roles.

Secondly, $\mathcal{SRIQ}$ is designed to be of similar practicability as $\mathcal{SHIQ}$. The tableau algorithm for $\mathcal{SHIQ}$ and the one for $\mathcal{SRIQ}$ presented here are very similar. Even though the additional expressive means of $\mathcal{SRIQ}$ require certain adjustments to the $\mathcal{SHIQ}$ algorithm, these adjustments do not add new sources of non-determinism, and, subject to empirical verification, are believed to be "harmless" in the sense of not significantly degrading typical performance as compared with the $\mathcal{SHIQ}$ algorithm. More precisely, we employ the same technique using finite automata as in (8) to handle role inclusions $R \circ S \dot{\sqsubseteq} R$ and $S \circ R \dot{\sqsubseteq} R$. This involves a pre-processing step which takes an Rbox and builds, for each role R, a finite automaton that accepts exactly those words $R_1 \dots R_n$ such that, in each model of the Rbox, $\langle x, y\rangle \in (R_1 \dots R_n)^{\mathcal{I}}$ implies $\langle x, y\rangle \in R^{\mathcal{I}}$. These automata are then used in the tableau expansion rules to check, for a node x with $\forall R.C \in \mathcal{L}(x)$ and an $R_1 \dots R_n$-neighbour y of x, whether to add C to $\mathcal{L}(y)$. Even though the pre-processing step might appear a little cumbersome, the usage of the automata in the algorithm makes it quite elegant and compact.

The current paper describes work in progress towards a description logic that overcomes certain shortcomings in expressiveness of other DLs. We have used $\mathcal{SHIN}$, $\mathcal{SHIQ}$, and $\mathcal{RIQ}$ as a starting point, extended them with some "useful-yet-harmless" expressive means, and also extended the tableau algorithm accordingly. We wish to discuss this extension in case we have overlooked other "useful-yet-harmless" expressive means, and we plan to further extend $\mathcal{SRIQ}$: currently, various new operators are restricted to *simple* roles, and we have yet to establish which of these restrictions are necessary in order to preserve decid-

ability[1] or practicability. Moreover, we plan to extend $\mathcal{SRIQ}$ towards $\mathcal{SHOIQ}$ (9), i.e., to also include nominals.

For a full specification of the tableau algorithm and proofs, see (6).

2 The Logic $\mathcal{SRIQ}$

In this section, we introduce the DL $\mathcal{SRIQ}$. This includes the definition of syntax, semantics, and inference problems.

2.1 Roles, Role Hierarchies, and Role Assertions

Definition 1 (Interpretations). *Let* $\mathbf{C}$ *be a set of* ***concept names****,* $\mathbf{R}$ *a set of* ***role names****, and* $\mathbf{I} = \{a, b, c \ldots\}$ *a set of* ***individual names****. The set of* ***roles*** *is* $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$*, where a role* R^- *is called the* ***inverse role*** *of* R*.*

As usual, an ***interpretation*** $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ *consists of a set* $\Delta^\mathcal{I}$*, called the* ***domain*** *of* $\mathcal{I}$*, and a* ***valuation*** $\cdot^\mathcal{I}$ *which associates, with each role name* R*, a binary relation* $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$*, with each concept name* C *a subset* $C^\mathcal{I} \subseteq \Delta^\mathcal{I}$ *and, with each individual name* a *an element* $a^\mathcal{I} \in \Delta^\mathcal{I}$*. Inverse roles are interpreted as usual, i.e., for each role* $R \in \mathbf{R}$*, we have*

$$(R^-)^\mathcal{I} = \{\langle y, x\rangle \mid \langle x, y\rangle \in R^\mathcal{I}\}.$$

Note that, unlike in the case of $\mathcal{SHIQ}$, we did not introduce *transitive role names*. This is so since, as will become apparent below, role box assertions can be used to force roles to be transitive.

To avoid considering roles such as R^{--}, we define a function Inv on roles such that $\mathsf{Inv}(R) = R^-$ if $R \in \mathbf{R}$ is a role name, and $\mathsf{Inv}(R) = S \in \mathbf{R}$ if $R = S^-$.

Since we will often work with a string of roles, it is convenient to extend both $\cdot^\mathcal{I}$ and $\mathsf{Inv}(\cdot)$ to such strings: if $w = R_1 \ldots R_n$ for R_i roles, then we set $w^\mathcal{I} = R_1^\mathcal{I} \circ \ldots \circ R_n^\mathcal{I}$ and $\mathsf{Inv}(w) = \mathsf{Inv}(R_n) \ldots \mathsf{Inv}(R_1)$, where $\circ$ denotes composition of binary relations.

A *role box* $\mathcal{R}$ consists of two components. The first component is a *role hierarchy* $\mathcal{R}_h$ which consists of (generalised) *role inclusion axioms*, i.e., statements of the form $R \dot{\sqsubseteq} S$, $RS \dot{\sqsubseteq} S$, and $SR \dot{\sqsubseteq} S$. The second component is a set $\mathcal{R}_a$ of *role assertions* stating, for instance, that a role R must be interpreted as a transitive, reflexive, irreflexive, symmetric, or transitive relation, or that two (possibly inverse) roles R and S are to be interpreted as *disjoint* binary relations.

We start with the definition of a role hierarchy, whose definition involves a strict partial order $\prec$ on roles, i.e., an irreflexive and transitive relation on $R \cup \{R^- \mid R \in \mathbf{R}\}$.

Definition 2 ((Regular) Role Inclusion Axioms).

Let $\prec$ *be a strict partial order on roles. A* ***role inclusion axiom*** *(RIA for short) is an expression of the form* $w \dot{\sqsubseteq} R$*, where* w *is a finite string of roles, and* R *is a role name. A* ***role hierarchy*** $\mathcal{R}_h$*, then, is a finite set of RIAs.*

[1] See (10) for such a case.

*An interpretation $\mathcal{I}$ **satisfies** a role inclusion axiom $S_1 \dots S_n \dot{\sqsubseteq} R$, if*

$$S_1^{\mathcal{I}} \circ \dots \circ S_n^{\mathcal{I}} \subseteq R^{\mathcal{I}},$$

*where $\circ$ stands for the composition of binary relations. An interpretation is a **model** of a role hierarchy R_h, if it satisfies all RIAs in R_h, written $\mathcal{I} \models R_h$. A RIA $w \dot{\sqsubseteq} R$ is $\prec$**-regular** if*

- *R is a role name,*
- *$w = RR$,*
- *$w = R^-$,*
- *$w = S_1 \dots S_n$ and $S_i \prec R$, for all $1 \leq i \leq n$,*
- *$w = RS_1 \dots S_n$ and $S_i \prec R$, for all $1 \leq i \leq n$, or*
- *$w = S_1 \dots S_n R$ and $S_i \prec R$, for all $1 \leq i \leq n$.*

*Finally, a role hierarchy $\mathcal{R}_h$ is said to be **regular** if there exists a strict partial order $\prec$ on roles such that each RIA in $\mathcal{R}_h$ is $\prec$-regular.*

Regularity prevents a role hierarchy from containing cyclic dependencies. For instance, the role hierarchy

$$\{RS \dot{\sqsubseteq} S, \quad RT \dot{\sqsubseteq} R, \quad UT \dot{\sqsubseteq} T, \quad US \dot{\sqsubseteq} U\}$$

is not regular because it would require $\prec$ to satisfy $S \prec U \prec T \prec R \prec S$, which would imply $S \prec S$, thus contradicting irreflexivity. Such cyclic dependencies are known to lead to undecidability (8).

From the definition of the semantics of inverse roles, it follows immediately that

$$\langle x, y \rangle \in w^{\mathcal{I}} \text{ iff } \langle y, x \rangle \in \mathsf{Inv}(w)^{\mathcal{I}}.$$

Hence, each model satisfying $w \dot{\sqsubseteq} S$ also satisfies $\mathsf{Inv}(w) \dot{\sqsubseteq} \mathsf{Inv}(S)$ (and vice versa), and thus the restriction to those RIAs with role *names* on their right hand side does not have any effect on expressivity.

Given a role hierarchy $\mathcal{R}_h$, we define the relation $\dot{\sqsubseteq}^*$ to be the transitive-reflexive closure of $\dot{\sqsubseteq}$ over $\{R \dot{\sqsubseteq} S, \mathsf{Inv}(R) \dot{\sqsubseteq} \mathsf{Inv}(S) \mid R \dot{\sqsubseteq} S \in \mathcal{R}_h\}$. A role R is called a **sub-role** (resp. **super-role**) of a role S if $R \dot{\sqsubseteq}^* S$ (resp. $S \dot{\sqsubseteq}^* R$). Two roles R and S are **equivalent** ($R \equiv S$) if $R \dot{\sqsubseteq}^* S$ and $S \dot{\sqsubseteq}^* R$.

Note that, due to the fourth restriction in the definition of $\prec$-regularity, we also restrict $\dot{\sqsubseteq}^*$ to be acyclic, and thus regular role hierarchies never contain two equivalent roles.[2]

Next, let us turn to the second component of Rboxes, the role assertions. For an interpretation $\mathcal{I}$, we define $Diag^{\mathcal{I}}$ to be the set $\{\langle x, x \rangle \mid x \in \Delta^{\mathcal{I}}\}$ and set $R^{\mathcal{I}} \downarrow := \{\langle x, x \rangle \mid \exists y \in \Delta^{\mathcal{I}}. \langle x, y \rangle \in R^{\mathcal{I}}\}$.

[2] This is not a serious restriction for, if $\mathcal{R}$ contains $\dot{\sqsubseteq}^*$ cycles, we can simply choose one role R from each cycle and replace all other roles in this cycle with R in the input Rbox, Tbox and Abox (see below).

Definition 3 (Role Assertions). *For roles R and S, we call the assertions* $\mathsf{Ref}(R)$, $\mathsf{Irr}(R)$, $\mathsf{Sym}(R)$, $\mathsf{Tra}(R)$, *and* $\mathsf{Dis}(R,S)$, ***role assertions****, where, for each interpretation $\mathcal{I}$ and all $x,\ y,\ z \in \Delta^{\mathcal{I}}$, we have:*

$\mathcal{I} \models \mathsf{Sym}(R)$ *if* $\langle x,y\rangle \in R^{\mathcal{I}}$ *implies* $\langle y,x\rangle \in R^{I}$;
$\mathcal{I} \models \mathsf{Tra}(R)$ *if* $\langle x,y\rangle \in R^{\mathcal{I}}$ *and* $\langle y,z\rangle \in R^{\mathcal{I}}$ *imply* $\langle x,z\rangle \in R^{I}$;
$\mathcal{I} \models \mathsf{Ref}(R)$ *if* $R^{\mathcal{I}} \downarrow \subseteq R^{\mathcal{I}}$;
$\mathcal{I} \models \mathsf{Irr}(R)$ *if* $R^{\mathcal{I}} \cap Diag^{\mathcal{I}} = \emptyset$;
$\mathcal{I} \models \mathsf{Dis}(R,S)$ *if* $R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$.

Adding symmetric and transitive role assertions is a trivial move since both of these expressive means can be replaced by complex role inclusion axioms as follows: for the role assertion $\mathsf{Sym}(R)$ we can add to the Rbox, equivalently, the role inclusion axiom $R^- \dot{\sqsubseteq} R$, and, for the role assertion $\mathsf{Tra}(R)$, we can add to the Rbox, equivalently, $RR \dot{\sqsubseteq} R$. The proof of this should be obvious. Thus, as far as expressivity is concerned, we can assume for convenience that no role assertions of the form $\mathsf{Tra}(R)$ or $\mathsf{Sym}(R)$ appear in $\mathcal{R}_a$, but that transitive and symmetric roles will be handled by the RIAs alone.

The situation is different, however, for the other Rbox assertions. Neither reflexivity nor irreflexivity nor disjointness of roles can be enforced by role inclusion axioms. However, as we shall see later, reflexivity and irreflexivity of roles are closely related to the new concept $\exists R.\mathsf{Self}$.

In $\mathcal{SHIQ}$, the application of qualified number restrictions has to be restricted to certain roles, called *simple roles*, to preserve decidability (10). In the context of $\mathcal{SRIQ}$, the definition of *simple role* has to be slightly modified, and simple roles figure not only in qualified number restrictions, but in several other constructs as well. Intuitively, non-simple roles are those that are implied by the composition of roles.

Given a role hierarchy $\mathcal{R}_h$ and a set of role assertions $\mathcal{R}_a$ (without transitivity or symmetry assertions), the set of roles that are **simple in** $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$ is inductively defined as follows:

- a role name is simple if it does not occur on the right hand side of a RIA in $\mathcal{R}_h$,
- an inverse role R^- is simple if R is, and
- if R occurs on the right hand side of a RIA in $\mathcal{R}_h$, then R is simple if, for each $w \dot{\sqsubseteq} R \in \mathcal{R}_h$, $w = S$ for a simple role S.

A set of role assertions $\mathcal{R}_a$ is called **simple** if all roles R, S appearing in role assertions of the form $\mathsf{Ref}(R)$, $\mathsf{Irr}(R)$, or $\mathsf{Dis}(R,S)$ are simple in $\mathcal{R}$. If $\mathcal{R}$ is clear from the context, we often use "simple" instead of "simple in $\mathcal{R}$".

Definition 4 (Role Box). *A* ***$\mathcal{SRIQ}$-role box*** *(Rbox for short) is a set $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$, where $\mathcal{R}_h$ is a regular role hierarchy and $\mathcal{R}_a$ is a finite, simple set of role assertions.*

An interpretation ***satisfies a role box*** *$\mathcal{R}$ (written $\mathcal{I} \models \mathcal{R}$) if $\mathcal{I} \models R_h$ and $\mathcal{I} \models \phi$ for all role assertions $\phi \in R_a$. Such an interpretation is called a model of $\mathcal{R}$.*

2.2 Concepts and Inference Problems for $\mathcal{SRIQ}$

We are now ready to define the syntax and semantics of $\mathcal{SRIQ}$-concepts.

Definition 5 ($\mathcal{SRIQ}$ Concepts, Tboxes, and Aboxes). *The set of $\mathcal{SRIQ}$-**concepts** is the smallest set such that*

- *every concept name and $\top, \bot$ are concepts, and,*
- *if C, D are concepts, R is a role (possibly inverse), S is a simple role (possibly inverse), and n is a non-negative integer, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall R.C$, $\exists R.C$, $\exists S.\mathsf{Self}$, $(\geqslant nS.C)$, and $(\leqslant nS.C)$ are also concepts.*

*A **general concept inclusion axiom** (GCI) is an expression of the form $C \dot{\sqsubseteq} D$ for two $\mathcal{SRIQ}$-concepts C and D. A **Tbox** $\mathcal{T}$ is a finite set of GCIs.*

*An **individual assertion** is of one of the following forms: $a:C$, $(a,b):R$, $(a,b):\neg S$, or $a \neq b$, for $a, b \in \mathbf{I}$ (the set of individual names), a (possibly inverse) role R, a (possibly inverse) simple role S, and a $\mathcal{SRIQ}$-concept C. A $\mathcal{SRIQ}$-**Abox** $\mathcal{A}$ is a finite set of individual assertions.*

Note that number restrictions $(\geqslant nS.C)$ and $(\leqslant nS.C)$, the concept $\exists S.\mathsf{Self}$, and negated role assertions $(a,b):\neg S$, are all restricted to *simple* roles. In the case of number restrictions we mentioned the reason for this restriction already: without it, already the satisfiability problem of $\mathcal{SHIQ}$-concepts is undecidable (10), even for a logic without inverse roles and with only *unqualifying* number restrictions (these are number restrictions of the form $(\geqslant nR.\top)$ and $(\leqslant nR.\top)$). For $\mathcal{SRIQ}$ and the remaining restrictions to simple roles in concept expressions as well as role assertions, it is part of future work to determine which of these restrictions to simple roles are necessary in order to preserve decidability or practicability. For example, it should be possible to also allow non-simple roles in negated role assertions $(a,b):\neg R$ without losing decidability.

Note also that, in the definition of $\mathcal{SRIQ}$-Aboxes, we do not assume the unique name assumption (UNA) (which is commonly assumed in DLs (4)). Rather, by allowing inequalities between individuals in the Abox to be explicitly stated, we increase flexibility while, obviously, the UNA can be regained by explicitly stating $a \neq b$ for every pair $a, b \in \mathbf{I}$ of individuals. Moreover, notice that, in contrast to standard Aboxes, $\mathcal{SRIQ}$-Aboxes can also contain negated role assertions of the form $(a,b):\neg R$.

Definition 6 (Semantics and Inference Problems).

*Given an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, concepts C, D, roles R, S, and non-negative integers n, the **extension of complex concepts** is defined inductively by the following equations, where $\sharp M$ denotes the cardinality of a set M:*

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}, \quad \bot^{\mathcal{I}} = \emptyset, \quad (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \quad \textit{(top, bottom, negation)}$$

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, \quad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \quad \textit{(conjunction, disjunction)}$$

$$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \quad \textit{(exists restriction)}$$

$$(\exists R.\mathsf{Self})^{\mathcal{I}} = \{x \mid \langle x,x\rangle \in R^{\mathcal{I}}\} \quad \textit{($\exists R.\mathsf{Self}$-concepts)}$$

$$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\} \quad \textit{(value restriction)}$$

$$(\geqslant nR.C)^{\mathcal{I}} = \{x \mid \sharp\{y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geqslant n\} \quad \textit{(atleast restriction)}$$

$$(\leqslant nR.C)^{\mathcal{I}} = \{x \mid \sharp\{y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leqslant n\} \quad \textit{(atmost restriction)}$$

An interpretation $\mathcal{I}$ is a ***model of a Tbox*** $\mathcal{T}$ *(written $\mathcal{I} \models \mathcal{T}$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for each GCI $C \dot{\sqsubseteq} D$ in $\mathcal{T}$.*

A concept C is called ***satisfiable*** *iff there is an interpretation $\mathcal{I}$ with $C^{\mathcal{I}} \neq \emptyset$. A concept D* ***subsumes*** *a concept C (written $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each interpretation. Two concepts are* ***equivalent*** *(written $C \equiv D$) if they are mutually subsuming. The above inference problems can be defined w.r.t. a general role box $\mathcal{R}$ and/or a Tbox $\mathcal{T}$ in the usual way, i.e., by replacing* interpretation *with* model of $\mathcal{R}$ *and/or* $\mathcal{T}$*.*

For an interpretation $\mathcal{I}$, an element $x \in \Delta^{\mathcal{I}}$ is called an ***instance*** *of a concept C iff $x \in C^{\mathcal{I}}$.*

An interpretation $\mathcal{I}$ ***satisfies*** *(is a model of)* ***an Abox*** *$\mathcal{A}$ ($\mathcal{I} \models \mathcal{A}$) if for all individual assertions $\phi \in \mathcal{A}$ we have $\mathcal{I} \models \phi$, where*

$$\begin{array}{llll} \mathcal{I} \models a\!:\!C & \text{if} \quad a^{\mathcal{I}} \in C^{\mathcal{I}}; & \mathcal{I} \models a \neq b & \text{if} \quad a^{\mathcal{I}} \neq b^{\mathcal{I}}; \\ \mathcal{I} \models (a,b)\!:\!R & \text{if} \quad \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}; & \mathcal{I} \models (a,b)\!:\!\neg R & \text{if} \quad \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \notin R^{\mathcal{I}}. \end{array}$$

An Abox $\mathcal{A}$ is ***consistent*** *with respect to an Rbox $\mathcal{R}$ and a Tbox $\mathcal{T}$ if there is a model $\mathcal{I}$ for $\mathcal{R}$ and $\mathcal{T}$ such that $\mathcal{I} \models \mathcal{A}$.*

For DLs that are closed under negation, subsumption and (un)satisfiability of concepts can be mutually reduced: $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable, and C is unsatisfiable iff $C \sqsubseteq \bot$. Furthermore, a concept C is satisfiable iff the Abox $\{a\!:\!C\}$ is consistent.

It is straightforward to extend these reductions to Rboxes and Tboxes. In contrast, the reduction of inference problems w.r.t. a Tbox to pure concept inference problems (possibly w.r.t. a role hierarchy), deserves special care: in (2; 11; 3), the *internalisation* of GCIs is introduced, a technique that realises exactly this reduction. For $\mathcal{SRIQ}$, this technique can be modified accordingly.

Now, note also that, instead of having a role assertion $\mathsf{Ref}(R) \in \mathcal{R}_a$, we can add, equivalently, the GCI $\exists R.\top \dot{\sqsubseteq} \exists R.\mathsf{Self}$ to $\mathcal{T}$, which can in turn be internalised. Likewise, instead of asserting $\mathsf{Irr}(R)$, we can, equivalently, add the GCI $\top \dot{\sqsubseteq} \neg \exists R.\mathsf{Self}$. Thus, we arrive at the following theorem:

Theorem 1. *1. Satisfiability and subsumption of $\mathcal{SRIQ}$-concepts w.r.t. Tboxes and Rboxes are polynomially reducible to (un)satisfiability of $\mathcal{SRIQ}$-concepts w.r.t. Rboxes.*

2. Consistency of $\mathcal{SRIQ}$-Aboxes w.r.t. Tboxes and Rboxes is polynomially reducible to consistency of $\mathcal{SRIQ}$-Aboxes w.r.t. Rboxes.

3. W.l.o.g., we can assume that Rboxes do not contain role assertions of the form $\mathsf{Irr}(R)$, $\mathsf{Ref}(R)$, $\mathsf{Tra}(R)$, or $\mathsf{Sym}(R)$.

With Theorem 1, all standard inference problems for $\mathcal{SRIQ}$-concepts and Aboxes can be reduced to the problem of determining the consistency of a $\mathcal{SRIQ}$-Abox w.r.t. to an Rbox, where we can assume w.l.o.g. that all role assertions in the Rbox are of the form $\mathsf{Dis}(R, S)$—we call such an Rbox **reduced**.

3 $\mathcal{SRIQ}$ is Decidable

As we have just seen, we can restrict our attention to the consistency of Aboxes w.r.t. reduced Rboxes only. We have extended the tableau algorithm for $\mathcal{RIQ}$ to $\mathcal{SRIQ}$, and will spend the remainder of this paper on its description.

In a first step, the tableau algorithm takes a reduced Rbox $\mathcal{R}$ and an Abox $\mathcal{A}$ and builds, for each possibly inverse role R occurring in $\mathcal{R}$ or $\mathcal{A}$, a non-deterministic *finite automaton* $\mathcal{B}_R$. Intuitively, such an automaton is used to memorise the path between an object x that has to satisfy a concept of the form $\forall R.C$ and other objects, and then to determine which of these objects must satisfy C. The following proposition states that $\mathcal{B}_R$ indeed captures all implications between (paths of) roles and R that are consequences of the role hierarchy $\mathcal{R}_h$, where $L(\mathcal{B}_R)$ denotes the language (a set of strings of roles) accepted by $\mathcal{B}_R$.

Proposition 1. *$\mathcal{I}$ is a model of $\mathcal{R}_h$ if and only if, for each (possibly inverse) role R occurring in $\mathcal{R}_h$, each word $w \in L(\mathcal{B}_R)$, and each $\langle x, y\rangle \in w^{\mathcal{I}}$, we have $\langle x, y\rangle \in R^{\mathcal{I}}$.*

Since Aboxes usually involve several individuals with arbitrary role relationships between them, the completion algorithm presented works on *forests* rather than on *trees*. A forest is a collection of trees whose root nodes correspond to the individuals appearing in the input Abox and which form an arbitrarily connected graph according to the role assertions stated in the Abox. Similar as for $\mathcal{RIQ}$, we define a set $\mathsf{fclos}(\mathcal{A}, \mathcal{R})$ of "relevant sub-concepts" of those concepts occurring in $\mathcal{A}$; see (6) for details.

Definition 7. *A **completion forest** $\mathbf{F}$ for a $\mathcal{SRIQ}$-Abox $\mathcal{A}$ and an Rbox $\mathcal{R}$ is a collection of trees whose distinguished **root nodes** can be connected arbitrarily. Moreover, each node x is labelled with a set $\mathcal{L}(x) \subseteq \mathsf{fclos}(\mathcal{A}, \mathcal{R})$ and each edge $\langle x, y\rangle$ from a node x to its **successor** y is labelled with a non-empty set $\mathcal{L}(\langle x, y\rangle)$ of (possibly inverse and possibly negated) roles occurring in $\mathcal{A}$ and $\mathcal{R}$. Finally, completion forests come with an explicit **inequality relation** $\neq$ on nodes which is implicitly assumed to be symmetric.*

*Let x and y be nodes in $\mathbf{F}$ and R a role. If $R' \sqsubseteq^* R$ and $R' \in \mathcal{L}(\langle x, y\rangle)$, then y is called an R-**successor** of x.*

*If y is an R-successor of x or x is an $\mathsf{Inv}(R)$-successor of y, then y is called an R-**neighbour** of x. Moreover, a node x is a **neighbour** of y, if it is an R-neighbour for some role R. **Successors**, **predecessors**, **ancestors**, and **descendants** are defined as usual.*

For a role S, a concept C, and a node x in $\mathbf{F}$, we define $S^{\mathbf{F}}(x, C)$ by

$$S^{\mathbf{F}}(x, C) := \{y \mid y \text{ is an } S\text{-neighbour of } x \text{ and } C \in \mathcal{L}(y)\}.$$

*A node is **blocked** iff it is either directly or indirectly blocked. A node x is **directly blocked** iff none of its ancestors are blocked, and it has ancestors x', y and y' such that*

1. *none of x', y and y' is a root node,*

2. x *is a successor of* x' *and* y *is a successor of* y' and
3. $\mathcal{L}(x) = \mathcal{L}(y)$ *and* $\mathcal{L}(x') = \mathcal{L}(y')$ and
4. $\mathcal{L}(\langle x', x\rangle) = \mathcal{L}(\langle y', y\rangle)$.

In this case, we say that y ***blocks*** x.

A node y *is* ***indirectly blocked*** *if one of its ancestors is blocked.*

Given a non-empty $\mathcal{SRIQ}$*-Abox* $\mathcal{A}$ *and a reduced Rbox* $\mathcal{R}$*, the tableau algorithm is initialised with the completion forest* $\mathbf{F}_{\mathcal{A},\mathcal{R}}$ *defined as follows:*

- *for each individual* a *occurring in* $\mathcal{A}$, $\mathbf{F}_{\mathcal{A},\mathcal{R}}$ *contains a* ***root node*** x_a,
- *if* $(a,b):R \in \mathcal{A}$ *or* $(a,b):\neg R \in \mathcal{A}$, *then* $\mathbf{F}_{\mathcal{A},\mathcal{R}}$ *contains an edge* $\langle x_a, x_b\rangle$,
- *if* $a \neq b \in \mathcal{A}$, *then* $x_a \neq x_b$ *is in* $\mathbf{F}_{\mathcal{A},\mathcal{R}}$,
- $\mathcal{L}(x_a) := \{C \mid a:C \in \mathcal{A}\}$, *and*
- $\mathcal{L}(\langle x_a, x_b\rangle) := \{R \mid (a,b):R \in \mathcal{A}\} \cup \{\neg R \mid (a,b):\neg R \in \mathcal{A}\}$.

A completion forest $\mathbf{F}$ *is said to* ***contain a clash*** *if there are nodes* x *and* y *such that*

1. $\bot \in \mathcal{L}(x)$, *or*
2. *for some concept name* A, $\{A, \neg A\} \subseteq \mathcal{L}(x)$, *or*
3. x *is an* S*-neighbour of* x *and* $\neg\exists S.\mathsf{Self} \in \mathcal{L}(x)$, *or*
4. x *and* y *are root nodes,* y *is an* R*-neighbour of* x, *and* $\neg R \in \mathcal{L}(\langle x, y\rangle)$, *or*
5. *there is some* $\mathsf{Dis}(R,S) \in \mathcal{R}_a$ *and* y *is an* R*- and an* S*-neighbour of* x, *or*
6. *there is some concept* $(\leqslant n S.C) \in \mathcal{L}(x)$ *and* $\{y_0, \ldots, y_n\} \subseteq S^{\mathbf{F}}(x, C)$ *with* $y_i \neq y_j$ *for all* $0 \leq i < j \leq n$.

A completion forest that does not contain a clash is called ***clash-free****. A completion forest is* ***complete*** *if none of the rules from Figure 1 can be applied to it.*

When started with a non-empty Abox $\mathcal{A}$ *and a reduced Rbox* $\mathcal{R}$*, the tableau algorithm initialises* $\mathbf{F}_{\mathcal{A},\mathcal{R}}$ *and repeatedly applies the expansion rules from Figure 1 to it, stopping when a clash occurs, and applying the shrinking rules eagerly, i.e., the* $\leq$*- and the* $\leq_r$*-rule are applied with highest priority. The algorithm answers "*$\mathcal{A}$ *is satisfiable w.r.t.* $\mathcal{R}$*" iff the expansion rules can be applied in such a way that they yield a complete and clash-free completion forest, and "*$\mathcal{A}$ *is unsatisfiable w.r.t.* $\mathcal{R}$*" otherwise.*

Lemma 1. *Let* $\mathcal{A}$ *be a* $\mathcal{SRIQ}$*-Abox where all concepts are in negation normal form and* $\mathcal{R}$ *a reduced Rbox.*

- *The tableau algorithm terminates when started for* $\mathcal{A}$ *and* $\mathcal{R}$.
- *The expansion rules can be applied to* $\mathcal{A}$ *and* $\mathcal{R}$ *such that they yield a complete and clash-free completion forest iff there is a tableau for* $\mathcal{A}$ *w.r.t.* $\mathcal{R}$.

From Theorem 1 and Lemma 1, we thus have the following theorem:

Theorem 2. *The tableau algorithm decides satisfiability and subsumption of* $\mathcal{SRIQ}$*-concepts with respect to Aboxes, Rboxes, and Tboxes.*

$\sqcap$-rule: if $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$,
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$

$\sqcup$-rule: if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and
$\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{E\}$ for some $E \in \{C_1, C_2\}$

$\exists$-rule: if $\exists S.C \in \mathcal{L}(x)$, x is not blocked, and
x has no S-neighbour y with $C \in \mathcal{L}(y)$
then create a new node y with
$\mathcal{L}(\langle x, y\rangle) := \{S\}$ and $\mathcal{L}(y) := \{C\}$

Self-rule: if $\exists S.\mathsf{Self} \in \mathcal{L}(x)$, x is not blocked, and $S \notin \mathcal{L}(\langle x, x\rangle)$
then add an edge $\langle x, x\rangle$ if it does not yet exist, and
set $\mathcal{L}(\langle x, x\rangle) \longrightarrow \mathcal{L}(\langle x, x\rangle) \cup \{S\}$

$\forall_1$-rule: if $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked, and
$\forall \mathcal{B}_S.C \notin \mathcal{L}(x)$
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{\forall \mathcal{B}_S.C\}$

$\forall_2$-rule: if $\forall \mathcal{B}(p).C \in \mathcal{L}(x)$, x is not indirectly blocked, $p \xrightarrow{S} q$ in $\mathcal{B}(p)$,
and there is an S-neighbour y of x with $\forall \mathcal{B}(q).C \notin \mathcal{L}(y)$,
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{\forall \mathcal{B}(q).C\}$

$\forall_3$-rule: if $\forall \mathcal{B}.C \in \mathcal{L}(x)$, x is not indirectly blocked, $\varepsilon \in L(\mathcal{B})$, and $C \notin \mathcal{L}(x)$
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C\}$

choose-rule: if $(\leqslant n S.C) \in \mathcal{L}(x)$, x is not indirectly blocked, and
there is an S-neighbour y of x with $\{C, \doteq\neg C\} \cap \mathcal{L}(y) = \emptyset$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{E\}$ for some $E \in \{C, \doteq\neg C\}$

$\geqslant$-rule: if $(\geqslant n S.C) \in \mathcal{L}(x)$, x is not blocked, and
there are no $y_1, \ldots, y_n \in S^{\mathbf{F}}(x, C)$
with $y_i \neq y_j$ for each $1 \leq i < j \leq n$
then create n new successors $y_1, \ldots, y_n$ of x with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$,
$\mathcal{L}(y_i) = \{C\}$, and $y_i \neq y_j$ for $1 \leq i < j \leq n$.

$\leqslant$-rule: if $(\leqslant n S.C) \in \mathcal{L}(x)$, x is not indirectly blocked, and
$\#S^{\mathbf{F}}(x, C) > n$, there are $y, z \in S^{\mathbf{F}}(x, C)$ with
not $y \neq z$ and y is not a root node nor an ancestor of z,
then 1. $\mathcal{L}(z) \longrightarrow \mathcal{L}(z) \cup \mathcal{L}(y)$ and
2. if z is an ancestor of x
then $\mathcal{L}(\langle z, x\rangle) \longrightarrow \mathcal{L}(\langle z, x\rangle) \cup \mathsf{Inv}(\mathcal{L}(\langle x, y\rangle))$
else $\mathcal{L}(\langle x, z\rangle) \longrightarrow \mathcal{L}(\langle x, z\rangle) \cup \mathcal{L}(\langle x, y\rangle)$
3. Set $u \neq z$ for all u with $u \neq y$.
4. remove y and the sub-tree below y from $\mathbf{F}$.

$\leqslant_r$-rule: if $(\leqslant n S.C) \in \mathcal{L}(x)$, $\#S^{\mathbf{F}}(x, C) > n$,
and there are two root nodes $y, z \in S^{\mathbf{F}}(x, C)$ with *not* $y \neq z$,
then 1. $\mathcal{L}(z) \longrightarrow \mathcal{L}(z) \cup \mathcal{L}(y)$ and
2. For all edges $\langle y, w\rangle$:
i. if the edge $\langle z, w\rangle$ does not exist, create it with $\mathcal{L}(\langle z, w\rangle) := \mathcal{L}(\langle y, w\rangle)$;
ii. else $\mathcal{L}(\langle z, w\rangle) \longrightarrow \mathcal{L}(\langle z, w\rangle) \cup \mathcal{L}(\langle y, w\rangle)$.
3. For all edges $\langle w, y\rangle$:
i. if the edge $\langle w, z\rangle$ does not exist, create it with $\mathcal{L}(\langle w, z\rangle) := \mathcal{L}(\langle w, y\rangle)$;
ii. else $\mathcal{L}(\langle w, z\rangle) \longrightarrow \mathcal{L}(\langle w, z\rangle) \cup \mathcal{L}(\langle w, y\rangle)$.
4. Set $u \neq z$ for all u with $u \neq y$.
5. Remove y and all incoming and outgoing edges from y from $\mathbf{F}$.

Fig. 1. The Expansion Rules for the $\mathcal{SRIQ}$ Tableau Algorithm.

Bibliography

[1] Areces, C., Blackburn, P., Hernandez, B., and Marx, M. Handling Boolean Aboxes. In *Proc. of the 2003 Description Logic Workshop (DL 2003)* (2003), CEUR (`http://ceur-ws.org/`).

[2] Baader, F. Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)* (Sydney, 1991).

[3] Baader, F., Bürckert, H.-J., Nebel, B., Nutt, W., and Smolka, G. On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations. *Journal of Logic, Language and Information 2* (1993), 1–18.

[4] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., Eds. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[5] Baader, F., Lutz, C., Milicic, M., Sattler, U., and Wolter, F. Integrating Description Logics and Action Formalisms: First Results. In *Proc. of the 20th National Conference on Artificial Intelligence (AAAI-05)* (2005), A. Press, Ed.

[6] Horrocks, I., Kutz, O., and Sattler, U. The Irresistible $\mathcal{SRIQ}$. Tech. rep., University of Manchester, 2005. Available at `http://www.cs.man.ac.uk/~sattler/publications/sriq-tr.pdf`.

[7] Horrocks, I., Patel-Schneider, P. F., and van Harmelen, F. From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language. *J. of Web Semantics 1*, 1 (2003), 7–26.

[8] Horrocks, I., and Sattler, U. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. *Artificial Intelligence 160* (2004), 79–104.

[9] Horrocks, I., and Sattler, U. A Tableaux Decision Procedure for $\mathcal{SHOIQ}$. In *Proc. of 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)* (2005), Morgan Kaufmann, Los Altos.

[10] Horrocks, I., Sattler, U., and Tobies, S. Practical Reasoning for Expressive Description Logics. In *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)* (1999), H. Ganzinger, D. McAllester, and A. Voronkov, Eds., vol. 1705 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 161–180.

[11] Schild, K. A Correspondence Theory for Terminological Logics: Preliminary Report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)* (Sydney, 1991), pp. 466–471.

[12] Wolstencroft, K., Brass, A., Horrocks, I., Lord, P., Sattler, U., Turi, D., and Stevens, R. A Little Semantic Web Goes a Long Way in Biology. In *Proc. of the 4th International Semantic Web Conference* (2005), LNCS, SV. To appear.

Appendix D:

"Description Logics in Ontology Applications" by Ian Horrocks, Proceedings of the 9th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2005), Koblenz, Germany, September 2005, Lecture Notes in Artificial Intelligence number 3702, pages 2-13, Springer, 2005. Enclosed with the full electronic version of this report as the file tableaux2005-ontologies.pdf.

Description Logics in Ontology Applications

Ian Horrocks

School of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, UK
horrocks@cs.man.ac.uk

Abstract. Description Logics (DLs) are a family of logic based knowledge representation formalisms. Although they have a range of applications (e.g., configuration and information integration), they are perhaps best known as the basis for widely used ontology languages such as OWL (now a W3C recommendation). This decision was motivated by a requirement that key inference problems be decidable, and that it should be possible to provide reasoning services to support ontology design and deployment. Such reasoning services are typically provided by highly optimised implementations of tableaux decision procedures; these have proved to be effective in applications in spite of the high worst case complexity of key inference problems. The increasing use of DL based ontologies in areas such as e-Science and the Semantic Web is, however, already stretching the capabilities of existing DL systems, and brings with it a range of research challenges.

1 Introduction

Description Logics (DLs) are a family of class (concept) based knowledge representation formalisms. They are characterised by the use of various constructors to build complex concepts from simpler ones, an emphasis on the decidability of key reasoning tasks, and by the provision of sound, complete and (empirically) tractable reasoning services.

Although they have a range of applications (e.g., reasoning with database schemas and queries [1–3]), DLs are perhaps best known as the basis for ontology languages such as OIL, DAML+OIL and OWL [4]. The decision to base these languages on DLs was motivated by a requirement not only that key inference problems (such as class satisfiability and subsumption) be decidable, but that "practical" decision procedures and "efficient" implemented systems also be available.

That DLs were able to meet the above requirements was the result of extensive research within the DL community over the course of the preceding 20 years or more. This research mapped out a complex landscape of languages, exploring a range of different language constructors, studying the effects of various combinations of these constructors on decidability and worst case complexity, and devising decision procedures, the latter often being tableaux based algorithms. At the same time, work on implementation and optimisation techniques demonstrated that, in spite of the high worst case complexity of key inference problems

(usually at least ExpTime), highly optimised DL systems were capable of providing practical reasoning support in the typical cases encountered in realistic applications.

With the added impetus provided by the OWL standardisation effort, DL systems are now being used to provide computational services for a rapidly expanding range of ontology tools and applications [5–10]. The increasing use of DL based ontologies in areas such as e-Science and the Semantic Web is, however, already stretching the capabilities of existing DL systems, and brings with it a range of research challenges.

2 Ontologies and Ontology Reasoning

In Computer Science, an ontology is usually taken to mean a conceptual model (of some domain), typically realised as a hierarchical vocabulary of terms, together with formal specifications of the meaning of each term. These specifications are often given with reference to other (simpler) terms in the ontology. For example, in a medical terminology ontology, the meaning of the term *Gastritis* might be specified as an *InflammatoryProcess* whose *outcome* is *InflammationOfStomach*, where *InflammatoryProcess*, *outcome* and *InflammationOfStomach* are all terms from the ontology. Such vocabularies may be used, e.g., to facilitate data sharing and reuse (often by annotating data using terms from a shared ontology), to structure data, or simply to explicate and investigate knowledge of a domain.

Ontologies play a major role in the Semantic Web (where they are used to annotate web resources) [11, 12], and are widely used in, e.g., knowledge management systems, e-Science, and bio-informatics and medical terminologies [13–16]. They are also of increasing importance in the Grid, where they may be used, e.g., to support the discovery, execution and monitoring of Grid services [17–19].

Given the formal and compositional nature of ontologies, it is natural to use logics as the basis for ontology languages—this allows for the precise definition of the meaning of compositional operators (such as "and" and "or"), and of relationships between terms (such as "subclass" and "instance"). The effective use of logic based ontology languages in applications will, however, critically depend on the provision of efficient reasoning support. On the one hand, such support is required by ontology engineers in order to help them to design and maintain sound, well-balanced ontologies [20]. On the other hand, such support is required by applications in order to exploit the formal specification of meaning captured in ontologies: querying ontologies and ontology structured data, is equivalent to computing logical entailments [21].

3 Ontology Languages and Description Logics

The OWL recommendation actually consists of three languages of increasing expressive power: OWL Lite, OWL DL and OWL Full. Like OWL's predecessor DAML+OIL, OWL Lite and OWL DL are basically very expressive description

logics with an RDF syntax. OWL Full provides a more complete integration with RDF, but its formal properties are less well understood, and key inference problems would certainly be *much* harder to compute.[1] For these reasons, OWL Full will not be considered here.

More precisely, OWL DL is based on the $\mathcal{SHOIQ}$ DL [23]; it restricts the form of number restrictions to be unqualified (see [24]), and adds a simple form of Datatypes (often called concrete domains in DLs [25]). Following the usual DL naming conventions, the resulting logic is called $\mathcal{SHOIN}(\mathbf{D})$, with the different letters in the name standing for (sets of) constructors available in the language: $\mathcal{S}$ stands for the basic $\mathcal{ALC}$ DL (equivalent to the propositional modal logic $\mathbf{K}_{(\mathbf{m})}$) extended with transitive roles [22], $\mathcal{H}$ stands for role hierarchies (equivalently, inclusion axioms between roles), $\mathcal{O}$ stands for nominals (classes whose extension is a single individual) [26], $\mathcal{N}$ stands for unqualified number restrictions and ($\mathbf{D}$) stands for datatypes) [27]. OWL Lite is equivalent to the slightly simpler $\mathcal{SHIF}(\mathbf{D})$ DL (i.e., $\mathcal{SHOIQ}$ without nominals, and with only functional number restrictions).

These equivalences allow OWL to exploit the considerable existing body of description logic research, e.g.:

- to define the semantics of the language and to understand its formal properties, in particular the decidability and complexity of key inference problems [28];
- as a source of sound and complete algorithms and optimised implementation techniques for deciding key inference problems [29, 22, 27];
- to use implemented DL systems in order to provide (partial) reasoning support [30–32].

3.1 $\mathcal{SHOIN}$ Syntax and Semantics

The syntax and semantics of $\mathcal{SHOIN}$ are briefly introduced here (we will ignore datatypes, as adding a datatype component would complicate the presentation and has little affect on reasoning [33]).

Definition 1. *Let* $\mathbf{R}$ *be a set of* role names *with both transitive and normal role names* $\mathbf{R}_+ \cup \mathbf{R}_\mathsf{P} = \mathbf{R}$*, where* $\mathbf{R}_\mathsf{P} \cap \mathbf{R}_+ = \emptyset$*. The set of* $\mathcal{SHOIN}$-roles *(or* roles *for short) is* $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$*. A* role inclusion axiom *is of the form* $R \sqsubseteq S$*, for two roles* R *and* S*. A* role hierarchy *is a finite set of role inclusion axioms.*

An interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ *consists of a non-empty set* $\Delta^\mathcal{I}$*, called the* domain *of* $\mathcal{I}$*, and a function* $\cdot^\mathcal{I}$ *which maps every role to a subset of* $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$ *such that, for* $P \in \mathbf{R}$ *and* $R \in \mathbf{R}_+$*,*

$$\langle x, y\rangle \in P^\mathcal{I} \text{ iff } \langle y, x\rangle \in P^{-\mathcal{I}},$$
$$\text{and if } \langle x, y\rangle \in R^\mathcal{I} \text{ and } \langle y, z\rangle \in R^\mathcal{I}, \text{ then } \langle x, z\rangle \in R^\mathcal{I}.$$

[1] Inference in OWL Full is clearly undecidable as OWL Full does not include restrictions on the use of transitive properties which are required in order to maintain decidability [22].

An interpretation $\mathcal{I}$ satisfies a role hierarchy $\mathcal{R}$ *iff* $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ *for each* $R \sqsubseteq S \in \mathcal{R}$*; such an interpretation is called a* model *of* $\mathcal{R}$*.*

Definition 2. *Let* N_C *be a set of* concept names *with a subset* $N_I \subseteq N_C$ *of* nominals*. The set of* $\mathcal{SHOIN}$-concepts *(or* concepts *for short) is the smallest set such that*

1. *every concept name* $C \in N_C$ *is a concept,*
2. *if* C *and* D *are concepts and* R *is a role, then* $(C \sqcap D)$*,* $(C \sqcup D)$*,* $(\neg C)$*,* $(\forall R.C)$*, and* $(\exists R.C)$ *are also concepts (the last two are called universal and existential restrictions, resp.), and*
3. *if* R *is a* simple *role*[2] *and* $n \in \mathbb{N}$*, then* $\leqslant nR$ *and* $\geqslant nR$ *are also concepts (called atmost and atleast number restrictions).*

The interpretation function $\cdot^{\mathcal{I}}$ *of an interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *maps, additionally, every concept to a subset of* $\Delta^{\mathcal{I}}$ *such that*

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \qquad \neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}},$$
$$\sharp o^{\mathcal{I}} = 1 \qquad \text{for all } o \in N_I,$$
$$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{There is a } y \in \Delta^{\mathcal{I}} \text{ with } \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\},$$
$$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{For all } y \in \Delta^{\mathcal{I}}, \text{ if } \langle x, y\rangle \in R^{\mathcal{I}}, \text{then } y \in C^{\mathcal{I}}\},$$
$$\leqslant nR^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\} \leqslant n\},$$
$$\geqslant nR^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\} \geqslant n\},$$

where, for a set M*, we denote the cardinality of* M *by* $\sharp M$*.*

For C *and* D *(possibly complex) concepts,* $C \dot{\sqsubseteq} D$ *is called a* general concept inclusion *(GCI), and a finite set of GCIs is called a* TBox*.*

An interpretation $\mathcal{I}$ satisfies *a GCI* $C \dot{\sqsubseteq} D$ *if* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$*, and* $\mathcal{I}$ *satisfies a TBox* $\mathcal{T}$ *if* $\mathcal{I}$ *satisfies each GCI in* $\mathcal{T}$*; such an interpretation is called a* model of $\mathcal{T}$*.*

A concept C *is called* satisfiable with respect to a role hierarchy $\mathcal{R}$ *and a TBox* $\mathcal{T}$ *if there is a model* $\mathcal{I}$ *of* $\mathcal{R}$ *and* $\mathcal{T}$ *with* $C^{\mathcal{I}} \neq \emptyset$*. Such an interpretation is called a* model of C *w.r.t.* $\mathcal{R}$ *and* $\mathcal{T}$*. A concept* D subsumes *a concept* C *w.r.t.* $\mathcal{R}$ *and* $\mathcal{T}$ *(written* $C \sqsubseteq_{\mathcal{R},\mathcal{T}} D$*) if* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ *holds in every model* $\mathcal{I}$ *of* $\mathcal{R}$ *and* $\mathcal{T}$*. Two concepts* C, D *are* equivalent *w.r.t.* $\mathcal{R}$ *and* $\mathcal{T}$ *(written* $C \equiv_{\mathcal{R},\mathcal{T}} D$*) iff they are mutually subsuming w.r.t.* $\mathcal{R}$ *and* $\mathcal{T}$*. (When* $\mathcal{R}$ *and* $\mathcal{T}$ *are obvious from the context, we will often write* $C \sqsubseteq D$ *and* $C \equiv D$*.) For an interpretation* $\mathcal{I}$*, an individual* $x \in \Delta^{\mathcal{I}}$ *is called an* instance *of a concept* C *iff* $x \in C^{\mathcal{I}}$*.*

Note that, as usual, subsumption and satisfiability can be reduced to each other, and reasoning w.r.t. *general TBoxes* and role hierarchies can be reduced to reasoning w.r.t. role hierarchies only [22, 27].

[2] A role is simple if it is neither transitive nor has any transitive subroles. Restricting number restrictions to simple roles is required in order to yield a decidable logic [22].

3.2 Practical Reasoning Services

Most modern DL systems use *tableaux* algorithms to test concept satisfiability. These algorithms work by trying to construct (a tree representation of) a model of the concept, starting from an individual instance. Tableaux expansion rules decompose concept expressions, add new individuals (e.g., as required by $\exists R.C$ terms),[3] and merge existing individuals (e.g., as required by $\leqslant nR.C$ terms). Nondeterminism (e.g., resulting from the expansion of disjunctions) is dealt with by searching the various possible models. For an unsatisfiable concept, all possible expansions will lead to the discovery of an obvious contradiction known as a *clash* (e.g., an individual that must be an instance of both A and $\neg A$ for some concept A); for a satisfiable concept, a complete and clash-free model will be constructed [34].

Tableaux algorithms have many advantages. It is relatively easy to design provably sound, complete and terminating algorithms, and the basic technique can be extended to deal with a wide range of class and role constructors. Moreover, although many algorithms have a higher worst case complexity than that of the underlying problem, they are usually quite efficient at solving the relatively easy problems that are typical of realistic applications.

Even in realistic applications, however, problems can occur that are much too hard to be solved by naive implementations of theoretical algorithms. Modern DL systems, therefore, include a wide range of optimisation techniques, the use of which has been shown to improve typical case performance by several orders of magnitude [29, 35, 36, 32, 37, 38]. Key techniques include lazy unfolding, absorption and dependency directed backtracking.

Lazy Unfolding In an ontology, or DL Tbox, large and complex concepts are seldom described monolithically, but are built up from a hierarchy of named concepts whose descriptions are less complex. The tableaux algorithm can take advantage of this structure by trying to find contradictions between concept names before adding expressions derived from Tbox axioms. This strategy is known as *lazy unfolding* [29, 36].

The benefits of lazy unfolding can be maximised by lexically *normalising* and *naming* all concept expressions and, recursively, their sub-expressions. An expression C is normalised by rewriting it in a standard form (e.g., disjunctions are rewritten as negated conjunctions); it is named by substituting it with a new concept name A, and adding an axiom $A \equiv C$ to the Tbox. The normalisation step allows lexically equivalent expressions to be recognised and identically named, and can even detect syntactically "obvious" satisfiability and unsatisfiability.

Absorption Not all axioms are amenable to lazy unfolding. In particular, so called *general concept inclusions* (GCIs), axioms of the form $C \sqsubseteq D$ where C is non-atomic, must be dealt with by explicitly making every individual in the

[3] Cycle detection techniques known as *blocking* may be required in order to guarantee termination.

model an instance of $D \sqcup \neg C$. Large numbers of GCIs result in a very high degree of non-determinism and catastrophic performance degradation [36].

Absorption is another rewriting technique that tries to reduce the number of GCIs in the Tbox by absorbing them into axioms of the form $A \sqsubseteq C$, where A is a concept name. The basic idea is that an axiom of the form $A \sqcap D \sqsubseteq D'$ can be rewritten as $A \sqsubseteq D' \sqcup \neg D$ and absorbed into an existing $A \sqsubseteq C$ axiom to give $A \sqsubseteq C \sqcap (D' \sqcup \neg D)$ [39]. Although the disjunction is still present, lazy unfolding ensures that it is only applied to individuals that are already known to be instances of A.

Dependency Directed Backtracking Inherent unsatisfiability concealed in sub-expressions can lead to large amounts of unproductive backtracking search known as thrashing. For example, expanding the expression $(C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \sqcap \exists R.(A \sqcap B) \sqcap \forall R.\neg A$ could lead to the fruitless exploration of 2^n possible expansions of $(C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n)$ before the inherent unsatisfiability of $\exists R.(A \sqcap B) \sqcap \forall R.\neg A$ is discovered. This problem is addressed by adapting a form of dependency directed backtracking called *backjumping*, which has been used in solving constraint satisfiability problems [40].

Backjumping works by labelling concepts with a dependency set indicating the non-deterministic expansion choices on which they depend. When a clash is discovered, the dependency sets of the clashing concepts can be used to identify the most recent non-deterministic expansion where an alternative choice might alleviate the cause of the clash. The algorithm can then jump back over intervening non-deterministic expansions *without* exploring any alternative choices. Similar techniques have been used in first order theorem provers, e.g., the "proof condensation" technique employed in the HARP theorem prover [41].

4 Research Challenges for Ontology Reasoning

The development of the OWL language, and the successful use of reasoning systems in tools such as the Protégé editor [42], has demonstrated the utility of logic and automated reasoning in the ontology domain. The increasing use of DL based ontologies in areas such as e-Science and the Semantic Web is, however, already stretching the capabilities of existing DL systems, and brings with it a range of challenges for future research.

Scalability Practical ontologies may be very large—tens or even hundreds of thousands of classes. Dealing with large-scale ontologies already presents a challenge to the current generation of DL reasoners, in spite of the fact that many existing large-scale ontologies are relatively simple. In the 40,000 concept Gene Ontology (GO), for example, much of the semantics is currently encoded in class names such as "heparin-metabolism"; enriching GO with more complex definitions, e.g., by explicitly modelling the fact that heparin-metabolism is a kind of "metabolism" that "acts-on" the carbohydrate "heparin", would make the semantics more accessible, and would greatly increase the value of GO by enabling

new kinds of query such as "what biological processes act on glycosaminoglycan" (heparin is a kind of glycosaminoglycan) [43]. However, adding more complex class definitions can cause the performance of existing reasoners to degrade to the point where it is no longer acceptable to users. Similar problems have been encountered with large medical terminology ontologies, such as the GALEN ontology [44].

Moreover, as well as using a conceptual model of the domain, many applications will also need to deal with very large volumes of instance data—the Gene Ontology, for example, is used to annotate millions of individuals, and practitioners want to answer queries that refer both to the ontology and to the relationships between these individuals, e.g., "what DNA binding products interact with insulin receptors". Answering this query requires a reasoner not only to identify individuals that are (perhaps only implicitly) instances of DNA binding products and of insulin receptors, but also to identify which pairs of individuals are (perhaps only implicitly) instances of the interactsWith role. For existing ontology languages it is possible to use DL reasoning to answer such queries, but dealing with the large volume of GO annotated gene product data is far beyond the capabilities of existing DL systems [45].

Several different approaches to this problem are already under investigation. One of these involves the use of a hybrid DL-DB architecture in which instance data is stored in a database, and query answering exploits the relatively simple relational structure encountered in typical data sets in order minimise the use of DL reasoning and maximise the use of database operations [46]. Another technique that is under investigation is to use reasoning techniques based on the encoding of $\mathcal{SHIQ}$ ontologies in Datalog [47]. On the one hand, theoretical investigations of this technique have revealed that data complexity (i.e., the complexity of answering queries against a fixed ontology and set of instance data) is significantly lower than the complexity of class consistency reasoning (i.e., NP-complete for $\mathcal{SHIQ}$, and even polynomial-time for a slight restriction of $\mathcal{SHIQ}$) [48]; on the other hand, the technique would allow relatively efficient Datalog engines to be used to store and reason with large volumes of instance data.

Expressive Power OWL is a relatively rich ontology language, but many applications require even greater expressive power than that which is provided by the existing OWL standard. For example, in ontologies describing complex physically structured domains such as biology [43] and medicine [44], it is often important to describe aggregation relationships between structures and their component parts, and to assert that certain properties of the component parts transfer to the structure as a whole (a femur with a fractured shaft is a fractured femur) [49]. The importance of this kind of knowledge can be gauged from the fact that various "work-arounds" have been described for use with ontology languages that cannot express it directly [50].

It may not be possible to satisfy all expressive requirements while staying within a decidable fragment of first order logic. Recent research has, therefore,

studied the use in ontology reasoning of semi-decision procedures such as resolution based theorem provers for full first order logic [51]. There have also been studies of languages that combine a DL with some other logical formalism, often Datalog style rules, with the connection between the two formalisms being restricted so as to maintain decidability [52, 47, 53]

Extended Reasoning Services Finally, in addition to solving problems of class consistency/subsumption and instance checking, explaining how such inferences are derived may be important, e.g., to help an ontology designer to rectify problems identified by reasoning support, or to explain to a user why an application behaved in an unexpected manner.

Work on developing practical explanation systems is at a relatively early stage, with different approaches still being developed and evaluated. One such technique involves exploiting standard reasoning services to identify a small set of axioms that still support the inference in question, the hope being that presenting a much smaller (than the complete ontology) set of axioms to the user will help them to understand the "cause" of the inference [54]. Another (possibly complementary) technique involves explaining the steps by which the inference was derived, e.g., using a sequence of simple natural deduction style inferences [55, 56].

As well as explanation, so-called "non-standard inferences" could also be important in supporting ontology design; these include matching, approximation, and difference computations. Non-standard inferences are the subject of ongoing research [57–60]; it is still not clear if they can be extended to deal with logics as expressive as those that underpin modern ontology languages, or if they will scale to large applications ontologies.

5 Summary

Description Logics are a family of class based knowledge representation formalisms characterised by the use of various constructors to build complex classes from simpler ones, and by an emphasis on the provision of sound, complete and (empirically) tractable reasoning services. They have been used in a wide range of applications, but perhaps most notably (at least in recent times) in providing a formal basis and reasoning services for (web) ontology languages such as OWL.

The effective use of logic based ontology languages in applications will, however, critically depend on the provision of efficient reasoning services to support both ontology design and deployment. The increasing use of DL based ontologies in areas such as e-Science and the Semantic Web is, however, already stretching the capabilities of existing DL systems, and brings with it a range of challenges for future research. The extended ontology languages needed in some applications may demand the use of more expressive DLs, and even for existing languages, providing efficient reasoning services is extremely challenging.

Some applications may even call for ontology languages based on larger fragments of FOL. The development of such languages, and reasoning services to

support them, extends these challenges to the whole logic based Knowledge Representation community.

Acknowledgements

I would like to acknowledge the contribution of those who provided me with inspiration and guidance, and the many collaborators with whom I have been privileged to work. These include Franz Baader, Sean Bechhofer, Dieter Fensel, Carole Goble, Frank van Harmelen, Carsten Lutz, Alan Rector, Ulrike Sattler, Peter F. Patel-Schneider, Stephan Tobies and Andrei Voronkov.

References

1. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Description logic framework for information integration. In: Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98). (1998) 2–13
2. Calvanese, D., De Giacomo, G., Lenzerini, M.: On the decidability of query containment under constraints. In: Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98). (1998) 149–158
3. Horrocks, I., Tessaris, S., Sattler, U., Tobies, S.: How to decide query containment under constraints using a description logic. In: Proc. of the 7th Int. Workshop on Knowledge Representation meets Databases (KRDB 2000), CEUR (http://ceur-ws.org/) (2000)
4. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. J. of Web Semantics **1** (2003) 7–26
5. Knublauch, H., Fergerson, R., Noy, N., Musen, M.: The protégé OWL plugin: An open development environment for semantic web applications. In McIlraith, S.A., Plexousakis, D., van Harmelen, F., eds.: Proc. of the 2004 International Semantic Web Conference (ISWC 2004). Number 3298 in Lecture Notes in Computer Science, Springer (2004) 229–243
6. Liebig, T., Noppens, O.: Ontotrack: Combining browsing and editing with reasoning and explaining for OWL Lite ontologies. In McIlraith, S.A., Plexousakis, D., van Harmelen, F., eds.: Proc. of the 2004 International Semantic Web Conference (ISWC 2004). Number 3298 in Lecture Notes in Computer Science, Springer (2004) 244–258
7. Rector, A.L., Nowlan, W.A., Glowinski, A.: Goals for concept representation in the GALEN project. In: Proc. of the 17th Annual Symposium on Computer Applications in Medical Care (SCAMC'93), Washington DC, USA (1993) 414–418
8. Visser, U., Stuckenschmidt, H., Schuster, G., Vögele, T.: Ontologies for geographic information processing. Computers in Geosciences (to appear)
9. Oberle, D., Sabou, M., Richards, D.: An ontology for semantic middleware: extending daml-s beyond web-services. In: Proceedings of ODBASE 2003. (2003)
10. Wroe, C., Goble, C.A., Roberts, A., Greenwood, M.: A suite of DAML+OIL ontologies to describe bioinformatics web services and data. Int. J. of Cooperative Information Systems (2003) Special Issue on Bioinformatics.
11. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic Web. Scientific American **284** (2001) 34–43

12. The DAML Services Coalition: DAML-S: Web service description for the semantic web. In: Proc. of the 2003 International Semantic Web Conference (ISWC 2003). Number 2870 in Lecture Notes in Computer Science, Springer (2003)
13. Uschold, M., King, M., Moralee, S., Zorgios, Y.: The enterprise ontology. Knowledge Engineering Review **13** (1998)
14. Stevens, R., Goble, C., Horrocks, I., Bechhofer, S.: Building a bioinformatics ontology using OIL. IEEE Transactions on Information Technology in Biomedicine **6** (2002) 135–141
15. Rector, A., Horrocks, I.: Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In: Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97), AAAI Press, Menlo Park, California (1997)
16. Spackman, K.: Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT. J. of the Amer. Med. Informatics Ass. (2000) Fall Symposium Special Issue.
17. Emmen, A.: The grid needs ontologies—onto-what? (2002) http://www.hoise.com/primeur/03/articles/monthly/AE-PR-02-03-7.html.
18. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Vanderbilt, P.: Grid service specification (draft). GWD-I draft , GGF Open Grid Services Infrastructure Working Group (2002) http://www.globalgridforum.org/.
19. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration (2002) http://www.globus.org/research/papers/ogsa.pdf.
20. Wolstencroft, K., McEntire, R., Stevens, R., Tabernero, L., Brass, A.: Constructing Ontology-Driven Protein Family Databases. Bioinformatics **21** (2005) 1685–1692
21. Horrocks, I., Tessaris, S.: Querying the semantic web: a formal approach. In Horrocks, I., Hendler, J., eds.: Proc. of the 2002 International Semantic Web Conference (ISWC 2002). Number 2342 in Lecture Notes in Computer Science, Springer-Verlag (2002) 177–191
22. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for expressive description logics. In Ganzinger, H., McAllester, D., Voronkov, A., eds.: Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99). Number 1705 in Lecture Notes in Artificial Intelligence, Springer (1999) 161–180
23. Horrocks, I., Sattler, U.: A tableaux decision procedure for $\mathcal{SHOIQ}$. In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005). (2005) To appear.
24. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2003)
25. Baader, F., Hanschke, P.: A schema for integrating concrete domains into concept languages. In: Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91). (1991) 452–457
26. Blackburn, P., Seligman, J.: Hybrid languages. J. of Logic, Language and Information **4** (1995) 251–272
27. Horrocks, I., Sattler, U.: Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In: Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001). (2001) 199–204
28. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W.: The complexity of concept languages. Information and Computation **134** (1997) 1–58
29. Baader, F., Franconi, E., Hollunder, B., Nebel, B., Profitlich, H.J.: An empirical analysis of optimization techniques for terminological representation systems or:

Making KRIS get a move on. Applied Artificial Intelligence. Special Issue on Knowledge Base Management **4** (1994) 109–132
30. Horrocks, I.: The FaCT system. In de Swart, H., ed.: Proc. of the 2nd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'98). Volume 1397 of Lecture Notes in Artificial Intelligence., Springer (1998) 307–312
31. Patel-Schneider, P.F.: DLP system description. In: Proc. of the 1998 Description Logic Workshop (DL'98), CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-11/ (1998) 87–89
32. Haarslev, V., Möller, R.: RACER system description. In: Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001). Volume 2083 of Lecture Notes in Artificial Intelligence., Springer (2001) 701–705
33. Pan, J.Z.: Description Logics: Reasoning Support for the Semantic Web. PhD thesis, University of Manchester (2004)
34. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for very expressive description logics. J. of the Interest Group in Pure and Applied Logic **8** (2000) 239–264
35. Bresciani, P., Franconi, E., Tessaris, S.: Implementing and testing expressive description logics: Preliminary report. In: Proc. of the 1995 Description Logic Workshop (DL'95). (1995) 131–139
36. Horrocks, I.: Using an expressive description logic: FaCT or fiction? In: Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98). (1998) 636–647
37. Patel-Schneider, P.F.: DLP. In: Proc. of the 1999 Description Logic Workshop (DL'99), CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-22/ (1999) 9–13
38. Horrocks, I., Patel-Schneider, P.F.: Optimizing description logic subsumption. J. of Logic and Computation **9** (1999) 267–293
39. Horrocks, I., Tobies, S.: Reasoning with axioms: Theory and practice. In: Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000). (2000) 285–296
40. Baker, A.B.: Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results. PhD thesis, University of Oregon (1995)
41. Oppacher, F., Suen, E.: HARP: A tableau-based theorem prover. J. of Automated Reasoning **4** (1988) 69–100
42. Protégé: http://protege.stanford.edu/ (2003)
43. Wroe, C., Stevens, R., Goble, C.A., Ashburner, M.: A methodology to migrate the Gene Ontology to a description logic environment using DAML+OIL. In: Proc. of the 8th Pacific Symposium on Biocomputing (PSB). (2003)
44. Rogers, J.E., Roberts, A., Solomon, W.D., van der Haring, E., Wroe, C.J., Zanstra, P.E., Rector, A.L.: GALEN ten years on: Tasks and supporting tools. In: Proc. of MEDINFO2001. (2001) 256–260
45. Horrocks, I., Li, L., Turi, D., Bechhofer, S.: The instance store: DL reasoning with large numbers of individuals. In: Proc. of the 2004 Description Logic Workshop (DL 2004). (2004) 31–40
46. Bechhofer, S., Horrocks, I., Turi, D.: The OWL instance store: System description. In: Proc. of the 20th Int. Conf. on Automated Deduction (CADE-20). Lecture Notes in Artificial Intelligence, Springer (2005) To appear.
47. Hustadt, U., Motik, B., Sattler, U.: Reducing SHIQ-description logic to disjunctive datalog programs. In: Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004). (2004) 152–162

48. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. In: Proc. of the 2004 International Semantic Web Conference (ISWC 2004). (2004) 549–563
49. Rector, A.: Analysis of propagation along transitive roles: Formalisation of the galen experience with medical ontologies. In: Proc. of DL 2002, CEUR (http://ceur-ws.org/) (2002)
50. Schulz, S., Hahn, U.: Parts, locations, and holes - formal reasoning about anatomical structures. In: Proc. of AIME 2001. Volume 2101 of Lecture Notes in Artificial Intelligence., Springer (2001)
51. Tsarkov, D., Riazanov, A., Bechhofer, S., Horrocks, I.: Using Vampire to reason with OWL. In McIlraith, S.A., Plexousakis, D., van Harmelen, F., eds.: Proc. of the 2004 International Semantic Web Conference (ISWC 2004). Number 3298 in Lecture Notes in Computer Science, Springer (2004) 471–485
52. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. In: Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004), Morgan Kaufmann, Los Altos (2004) 141–151
53. Rosati, R.: On the decidability and complexity of integrating ontologies and rules. J. of Web Semantics **3** (2005) 61–73
54. Schlobach, S., Cornet, R.: Explanation of terminological reason-ing: A preliminary report. In: Proc. of the 2003 Description Logic Workshop (DL 2003). (2003)
55. McGuinness, D.L.: Explaining Reasoning in Description Logics. PhD thesis, Rutgers, The State University of New Jersey (1996)
56. Borgida, A., Franconi, E., Horrocks, I.: Explaining $\mathcal{ALC}$ subsumption. In: Proc. of the 14th Eur. Conf. on Artificial Intelligence (ECAI 2000). (2000)
57. Baader, F., Küsters, R., Borgida, A., McGuinness, D.L.: Matching in description logics. J. of Logic and Computation **9** (1999) 411–447
58. Brandt, S., Turhan, A.Y.: Using non-standard inferences in description logics — what does it buy me? In: Proc. of KI-2001 Workshop on Applications of Description Logics (KIDLWS'01). Volume 44 of CEUR (http://ceur-ws.org/). (2001)
59. Küsters, R.: Non-Standard Inferences in Description Logics. Volume 2100 of Lecture Notes in Artificial Intelligence. Springer Verlag (2001)
60. Brandt, S., Küsters, R., Turhan, A.Y.: Approximation and difference in description logics. In: Proc. of the 8th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2002). (2002) 203–214

Appendix E:

"OWL 1.1 Web Ontology Language Syntax" by Peter F. Patel-Schneider, Bell Labs Research, Lucent Technologies, 12 January 2006. Enclosed with the full electronic version of this report as the file owl-1.1-syntax.html.

OWL 1.1 Web Ontology Language Syntax

Editor's Draft of 12 January 2006

This version:
http://www-db.research.bell-labs.com/user/pfps/owl/syntax-20060112.html/
Latest version:
http://www-db.research.bell-labs.com/user/pfps/owl/syntax.html/
Previous version:
http://www-db.research.bell-labs.com/user/pfps/owl/syntax-20051219.html/
Editor:
Peter F. Patel-Schneider, Bell Labs Research, Lucent Technologies

---

Abstract

OWL 1.1 extends OWL DL in several ways, extending the Description Logic underlying OWL 1.1 to SROIQ, adding user-defined datatypes and restrictions involving datatype predicates, adding a punning version of meta-modelling, and adding a semantics-free comment mechanism. This document defines the function-style syntax for OWL 1.1, and provides an informal discussion of the meaning of the additional constructs.

Status of this Document

This is a editor's draft, for comment by the OWL community.

When approved, this document will serve as a target for implementation by the major developers of OWL systems.

This document has been produced for approval of the ad-hoc OWL community. Comments, both from within and without the community, are welcome. Although the development of OWL 1.1 is not a W3C activity, public comments should be sent to the semantic-web@w3c.org (archive) to ensure widest visibility.

Table of Contents

---

1. OWL DL Syntax

This is a recapitulation of the OWL DL syntax, with some changes that do not affect the language, included here as a basis for the OWL 1.1 syntax. The OWL DL syntax is called an abstract syntax, which is not quite right, so the syntax is here called the function-style syntax where necessary.

The syntax is specified here by means of a version of Extended BNF, very similar to the EBNF notation used for XML [*XML*]. Terminals are quoted; non-terminals are bold and not quoted. Alternatives are either separated by vertical bars (|) or are given in different productions. Components that can occur at most once are enclosed in square brackets ([...]); components that can occur any number of times (including zero) are enclosed in braces ({...}). Whitespace is ignored in the productions.

1.0. Preliminaries

Names in the syntax are RDF URI references, [*RDF Concepts*]. These names can be abbreviated into qualified names. The following prefixes are often used:

Namespace name	Namespace

rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
xsd	http://www.w3.org/2001/XMLSchema#
owl	http://www.w3.org/2002/07/owl#

Names are divided up into several syntactic categories:

```
datatypeID ::= URIreference
classID ::= URIreference
individualID ::= URIreference
ontologyID ::= URIreference
datavaluedPropertyID ::= URIreference
individualvaluedPropertyID ::= URIreference
annotationPropertyID ::= URIreference
ontologyPropertyID ::= URIreference
```

A name cannot be both a datatypeID and a classID in an ontology. A name also cannot be more than one of an datavaluedPropertyID, an individualvaluedPropertyID, an annotationPropertyID, or an ontologyPropertyID in an ontology.

Further, a name cannot be more than one of the above categories in OWL DL. (This limitation is separated because it is relaxed in OWL 1.1.)

There are two built-in classes in OWL. The class with identifier owl:Thing is the class of all individuals. The class with identifier owl:Nothing is the empty class.

The following XML Schema datatypes [XML Schema Datatypes] can be used in OWL as built-in datatypes by means of the XML Schema canonical URI reference for the datatype: xsd:string, xsd:boolean, xsd:decimal, xsd:float, xsd:double, xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:integer, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte, and xsd:positiveInteger.

There are several built-in annotation properties in OWL, namely owl:versionInfo, rdfs:label, rdfs:comment, rdfs:seeAlso, and rdfs:isDefinedBy. In keeping with their definition in RDF, rdfs:label and rdfs:comment can only be used with data literals.

There are also several built-in ontology properties; they are owl:imports, owl:priorVersion, owl:backwardCompatibleWith, and owl:incompatibleWith. Ontology annotations that use owl:imports have the extra effect of importing the target ontology.

1.1. Ontologies and Annotations

```
ontology ::= 'Ontology(' [ ontologyID ] { directive } ')'
directive ::= 'Annotation(' ontologyPropertyID ontologyID ')'
            | 'Annotation(' annotationPropertyID URIreference ')'
            | 'Annotation(' annotationPropertyID dataLiteral ')'
            | 'Annotation(' annotationPropertyID individual ')'
            | axiom
            | fact

annotation ::= 'annotation(' annotationPropertyID URIreference ')'
             | 'annotation(' annotationPropertyID dataLiteral ')'
             | 'annotation(' annotationPropertyID individual ')'
```

1.2. Facts

```
fact ::= individual
individual ::= 'Individual(' [ individualID ] { annotation } { 'type(' type ')' } { value } ')'
value ::= 'value(' individualvaluedPropertyID individualID ')'
        | 'value(' individualvaluedPropertyID  individual ')'
        | 'value(' datavaluedPropertyID  dataLiteral ')'

type ::= description

dataLiteral ::= typedLiteral | plainLiteral>
typedLiteral ::= lexicalForm^^URIreference
plainLiteral ::= lexicalForm | lexicalForm@languageTag
lexicalForm ::= as in RDF, a unicode string in normal form C
languageTag ::= as in RDF, an XML language tag

fact ::= 'SameIndividual(' individualID individualID {individualID} ')'
       | 'DifferentIndividuals(' individualID individualID {individualID} ')'
```

1.3. Axioms

To preserve decidability of reasoning in OWL Lite, not all properties can have cardinality restrictions placed on them or be specified as functional or inverse-functional. An individual-valued property is *complex* if 1/ it is specified as being functional or inverse-functional, 2/ there is some cardinality restriction that uses it, 3/ it has an inverse that is complex, or 4/ it has a super-property that is complex. Individual-valued properties that are not complex are called *simple*. Only simple properties can be specified as being transitive.

1.3.1. Class Axioms

```
axiom ::= 'Class(' classID  ['Deprecated'] modality { annotation } { description } ')'
```

```
modality ::= 'complete' | 'partial'

axiom ::= 'EnumeratedClass(' classID ['Deprecated'] { annotation } { individualID } ')'

axiom ::= 'DisjointClasses(' description description { description } ')'
        | 'EquivalentClasses(' description { description } ')'
        | 'SubClassOf(' description description ')'

axiom ::= 'Datatype(' datatypeID ['Deprecated']  { annotation } )'
```

1.3.2. Descriptions

```
description ::= classID
              | restriction
              | 'unionOf(' { description } ')'
              | 'intersectionOf(' { description } ')'
              | 'complementOf(' description ')'
              | 'oneOf(' { individualID } ')'
```

1.3.3. Restrictions

```
restriction ::= 'restriction(' datavaluedPropertyID dataRestrictionComponent { dataRestrictionComponent } ')'
              | 'restriction(' individualvaluedPropertyID individualRestrictionComponent { individualRestrictionComp
dataRestrictionComponent ::= 'allValuesFrom(' dataRange ')'
              | 'someValuesFrom(' dataRange ')'
              | 'value(' dataLiteral ')'
              | dataCardinality
dataCardinality ::= 'minCardinality(' non-negative-integer ')'
              | 'maxCardinality(' non-negative-integer ')'
              | 'cardinality(' non-negative-integer ')'
individualRestrictionComponent ::= 'allValuesFrom(' description ')'
              | 'someValuesFrom(' description ')'
              | 'value(' individualID ')'
              | individualCardinality
individualCardinality ::= 'minCardinality(' non-negative-integer ')'
              | 'maxCardinality(' non-negative-integer ')'
              | 'cardinality(' non-negative-integer ')'
dataRange ::= datatypeID | 'rdfs:Literal'
            | 'oneOf(' { dataLiteral } ')'
```

1.3.4. Property Axioms

```
axiom ::= 'DatatypeProperty(' datavaluedPropertyID ['Deprecated'] { annotation }
                { 'super(' datavaluedPropertyID ')'} ['Functional']
                { 'domain(' description ')' } { 'range(' dataRange ')' } ')'
        | 'ObjectProperty(' individualvaluedPropertyID ['Deprecated'] { annotation }
                { 'super(' individualvaluedPropertyID ')' }
                [ 'inverseOf(' individualvaluedPropertyID ')' ] [ 'Symmetric' ]
                { individualvaluedPropertyFlags }
                { 'domain(' description ')' } { 'range(' description ')' } ')'
        | 'AnnotationProperty(' annotationPropertyID { annotation } ')'
        | 'OntologyProperty(' ontologyPropertyID { annotation } ')'

individualvaluedPropertyFlags ::= 'Functional' | 'InverseFunctional' | 'Transitive'

axiom ::= 'EquivalentProperties(' datavaluedPropertyID datavaluedPropertyID  { datavaluedPropertyID } ')'
        | 'SubPropertyOf(' datavaluedPropertyID  datavaluedPropertyID ')'
        | 'EquivalentProperties(' individualvaluedPropertyID individualvaluedPropertyID
                                  { individualvaluedPropertyID } ')'
        | 'SubPropertyOf(' individualvaluedPropertyID  individualvaluedPropertyID ')'
```

2. OWL 1.1 Syntax Additions

2.1. Syntactic Sugar

OWL 1.1 extendes the syntax of OWL in two ways that simply provide shorthand notations (syntactic sugar) for expressive power otherwise in OWL DL.

```
axiom ::= 'DisjointUnion(' description description { description } ')'

value ::= 'valueNot(' individualvaluedPropertyID individualID ')'
        | 'valueNot(' individualvaluedPropertyID  individual ')'
        | 'valueNot(' datavaluedPropertyID  dataLiteral ')'
```

The first construct is simply the obvious combination of a DisjointClasses of all the descriptions except the first and an EquivalentClasses of the first description and the union of the other descriptions. The second construct is the complementOf the restriction of the property to the value.

2.2. SROIQ

OWL 1.1 includes a number of Description Logic constructs that raise the core Description Logic expressive power from SHOIN to SROIQ [*SROIQ*]. This amounts to adding qualified cardinality restrictions; local reflexivity restrictions for simple properties; reflexive, irreflexive, and anti-symmetric flags for simple properties; disjointness of simple properties; and regular property inclusion axioms.

```
dataCardinality ::= 'minCardinality(' non-negative-integer dataRange ')'
              | 'maxCardinality(' non-negative-integer dataRange ')'
```

```
                | 'cardinality(' non-negative-integer dataRange ')'

individualCardinality ::= 'minCardinality(' non-negative-integer description ')'
                | 'maxCardinality(' non-negative-integer description ')'
                | 'cardinality(' non-negative-integer description ')'

individualRestrictionComponent ::= 'self'

individualvaluedPropertyFlags ::= 'Reflexive' | 'Irreflexive' | 'Symmetric' | 'AntiSymmetric'

axiom ::= 'DisjointProperties(' dataavaluedPropertyID datavaluedPropertyID  { datavaluedPropertyID } ')'
        | 'DisjointProperties(' individualvaluedPropertyID individualvaluedPropertyID  { individualvaluedProperty

axiom ::= 'SubPropertyOf(propertyChain(' individualvaluedPropertyID { individualvaluedPropertyID } ')'
                                                                        individualvaluedPropertyID ')'
```

Only simple properties can have the self restriction component. Only simple properties can be specified as being Reflexive, Irreflexive, Symmetric, or Antisymmetric. Only simple properties can used in DisjointProperties axioms for individual-valued properties.

The SubPropertyOf axioms involving individual-valued properties must be *regular*. That is, there must be a strict partial order, <, on individual-valued properties such that for each SubPropertyOf axiom involving individual-valued properties, of the form SubPropertyOf(S R)

1. S is the inverse of R,
2. S is of the form propertyChain(R ... R),
3. S is of the form propertyChain(S1 ... Sn) and each Si \lt; R,
4. S is of the form propertyChain(R S1 ... Sn) and each Si \lt; R, or
5. S is of the form propertyChain(S1 ... Sn R) and each Si \lt; R.

The meaning of all these constructs is the same as in SROIQ.

2.3. Datatypes

OWL 1.1 includes its own methods for user-defined datatypes. The syntax for OWL 1.1 user-defined datatypes is similar to the one used in Protege. The semantics for OWL 1.1 user-defined datatypes is taken from XML Schema Datatypes [*XML Schema Datatypes*].

```
dataRange ::= 'datatype(' datatypeID { datatypeRestriction } ')'
datatypeRestriction ::= datatypeFacet'(' dataLiteral')'
datatypeFacet ::= 'length' | 'minLength' | 'maxLength' | 'pattern' | 'enumeration'
                | 'maxInclusive' | 'maxExclusive' | 'minInclusive' | 'minExclusive'
                | 'totalDigits' | 'fractionDigits'

axiom ::= 'Datatype(' datatypeID { annotation } 'base(' datatypeID ')' { datatypeRestriction } ')'
```

Datatype facets should only by used where they would be allowed in XML Schema Datatypes [*XML Schema Datatypes*], except that the 'length', 'minLength', 'maxLength', and 'pattern' facets are not allowed for numeric types. Datatype facets have the same meaning as in XML Schema Datatypes, except that they uniformly work in the value space, never the lexical space. If a datatype facet is used in a way that has no meaning, such as (length "5"^^xsd:string) or (datatype xsd:string (maxInclusive "5"^^xsd:int)) then the datatype extension is empty. Note that this means that rdfs:Literal has no useful facets.

OWL 1.1 allows restrictions that relate values for different data-valued properties on the same individual.

```
restriction ::= 'holds(' datatypePredicateID { argument } ')'
restriction ::= datatypePropertyID | dataLiteral
datatypePredicatesID ::= 'equal' | 'notEqual' | 'lessThan' | 'lessThanEqual' | 'greaterThan' | 'greaterThanEqual'
```

The syntax here allows an arbitrary number of arguments, but must be appropriate for the predicate, and all the current predicates only allow two arguments.

The base types of the values being compared must be the same, although this may be liberalized if the Semantic Web Best Practices note on datatypes is generally adopted. All invalid combinations are unsatisfiable (i.e., they do not signal an error). The equality and order for a particular base type is taken from XML Schema Datatypes [*XML Schema Datatypes*]. If a base datatype does not have an order then the restriction is unsatisfied.

2.4. Meta-modelling and annotations

OWL 1.1 removes the limitation on names being only one of a class, a property, or an individual in OWL DL. The semantic change to allow this without computational consequences is to break the RDF-inspired connection between class and property extensions and the individual denotation of names.

With this change, non-annotation properties can be placed on any name. The property applies to the use of the name as an individual. As a simple syntactic sugar, non-annotation properties can be part of certain class and property axioms.

OWL 1.1 changes the status of the built-in properties rdfs:label and rdfs:comment from annotation properties to data-valued properties with no domain or range.

```
annotation ::= value
        | 'type(' description ')'
```

A class or property axiom with an annotation is syntactic sugar for an extra Individual axiom for the name with just the annotations.

2.5. Comments

OWL 1.1 allows arbitrary comments to be inserted in ontologies.

```
comment ::= 'Comment(' { commentEntry } ')'
commentEntry ::= dataLiteral | URIreference
```

A comment is allowed anywhere white space is allowed and acts as white space.

Comments have no semantic import in OWL 1.1, but systems are expected to keep track of comments.

References

[OWL S&AS]
OWL Web Ontology Language Semantics and Abstract Syntax. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. W3C Recommendation 10 February 2004. Latest version is available at http://www.w3.org/TR/owl-semantics/.

[SROIQ]
The Even More Irresistible SROIQ. Ian Horrocks, Oliver Kutz, and Uli Sattler. Technical report, University of Manchester, 2005.

[XML]
Extensible Markup Language (XML) 1.0 (Second Edition). Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler, eds. W3C Recommendation 6 October 2000. Latest version is available at http://www.w3.org/TR/REC-xml.

[XML Schema Datatypes]
XML Schema Part 2: Datatypes Second Edition.. Paul V. Biron and Ashok Malhotra, eds. W3C Recommendation 82 October 2004. Latest version is available at http://www.w3.org/TR/xmlschema-2/.

Appendix F:

"Optimised Classification for Taxonomic Knowledge Bases" by Dmitry Tsarkov and Ian Horrocks, in Proceedings of the 2005 Description Logic Workshop (DL-2005), Edinburgh, Scotland, July 2005. Enclosed with the full electronic version of this report as the file dl2005-optimised.pdf.

Optimised Classification for Taxonomic Knowledge Bases

Dmitry Tsarkov and Ian Horrocks
University of Manchester, Manchester, UK
{tsarkov|horrocks}@cs.man.ac.uk

Abstract

Many legacy ontologies are now being translated into Description Logic (DL) based ontology languages in order to take advantage of DL based tools and reasoning services. The resulting DL Knowledge Bases (KBs) are typically of large size, but have a very simple structure, i.e., they consist mainly of shallow taxonomies. The classification algorithms used in state-of-the-art DL reasoners may not deal well with such KBs In this paper we propose an optimisation which dramatically speeds-up classification for such KBs.

1 Introduction

Motivated by the W3C recommendation, and the availability of DL based tools and reasoning services, many legacy ontologies are now being translated into the Description Logic (DL) based OWL DL ontology language [3]. The resulting DL Knowledge Bases (KBs) are typically of (very) large size, but often have only a (very) simple structure. In particular, the sub-class hierarchy often resembles a taxonomy (i.e., an asserted hierarchy of primitive concepts), and is very broad and shallow.

The classification algorithms used in state-of-the-art DL reasoners (such as FaCT and RACER) may not deal well with such KBs, mainly due to the fact that some classes in the hierarchy will have very large numbers of direct sub-classes. For such KBs, the top-down phase of the standard classification algorithm [2] will perform a very large number of subsumption tests, almost all of which will fail (i.e., conclude that no subsumption relationship holds). Although the simple structure of the KB means that the time required for each such test is small, the very large number of tests can still lead to performance problems.

In this paper we present a classification optimisation that identifies (subsets of) KBs for which it is possible to compute the concept hierarchy without performing *any* subsumption tests. This technique is very effective when used on the kinds of legacy KB described above as it avoids performing large numbers

of negative subsumption tests. It also turns out to produce (smaller) improvements in classification times for some more complex ontologies, as even in these ontologies a significant part of the upper level structure often resembles a taxonomy.

The drawback of this new optimisation is that it is only applicable to purely "definitional" knowledge bases, i.e., those that do not contain any general concept inclusion axioms (GCIs). However, the well known *absorption* optimisation has already been shown to be able to eliminate GCIs from typical ontologies [6] and, moreover, when absorption fails to eliminate GCIs, we may already expect serious reasoning performance problems from other sources (i.e., the hardness of individual subsumption tests). It is also only applicable to that portion of a KB which has the necessary simple structure, but this is often a major part of legacy KBs and, as mentioned above, is usually a significant part of more complex KBs.

2 Preliminaries

Description Logics are concept (class) based knowledge representation systems. They are (usually) decidable fragments of First Order Predicate Calculus, and have a have standard first order style model theoretic semantics [1]. The formal specification of semantics coupled with decidability allows for the design of decision procedures (sound, complete and terminating algorithms) for key reasoning tasks such as concept subsumption.

A DL Knowledge Base is often thought of as consisting of two parts: a *Tbox* and an *Abox*. The Tbox consists of a set of axioms that describe constraints on instances of given concepts (roughly akin to a conceptual schema in a database setting); the Abox consists of a set of axioms that assert instance relationships between individuals and concepts, and role relationships between pairs of individuals (roughly akin to data in a database setting). Tbox axioms are of the form $C \sqsubseteq D$ or $C \equiv D$, where C and D are concepts. When C is a concept name, such an axiom is often called a *definition* (of C), and when a definition is of the form $C \sqsubseteq D$, C is called a *primitive* concept.

Classification of a Tbox is the task of computing and caching the concept hierarchy for all of the named concepts that occur in the Tbox, i.e., computing the subsumption partial ordering of the named concepts.[1] Tbox classification is a basic reasoning task for DL reasoners—the concept hierarchy may be interesting in its own right, and is used to optimise many other reasoning tasks (e.g., query answering). This latter point is particularly important as, even if the Tbox part of a KB is very simple, the Abox may describe instances of complex concept expressions. This is typical, e.g., for applications of the Gene Ontology [4].

We will first briefly recall the optimised procedure for computing the concept

[1] We assume w.l.o.g. that all concept names occurring in the Abox also occur in the Tbox; this can be achieved by adding axioms of the form $C \sqsubseteq \top$ to the Tbox for any concept name C that would otherwise occur only in the Abox.

hierarchy first described in [2]. In this procedure, all concept names are sorted into *definitional order*, i.e. if concept name D occurs in the definition of concept name C, then $D \leq C$.[2] The concept hierarchy is initialised to contain the two concepts $\top$ and $\bot$ (with $\top$ being a super concept of $\bot$), and the named concepts are classified (added to the hierarchy in the appropriate position) one at a time in definitional order.

Classifying a concept involves two phases: a top-down phase in which its *parents* (i.e., direct subsumers) are computed, and a bottom-up phase in which its *children* (i.e., direct subsumees) are computed. In many cases, adding concepts in definitional order may allow the bottom-up phase to be omitted (because when a new concept is classified its only child will be $\bot$).

Various optimisations are used in order to minimise the number of subsumption tests needed in each phase. For example, when adding a concept C to the hierarchy, a top-down breadth first traversal is used that only checks if D subsumes C when it has already been determined that C is subsumed by all the concepts in the hierarchy that subsume D. The structure of Tbox axioms is also used to compute a set of *told subsumers* of C (i.e., trivially obvious subsumers). For example, if the Tbox contains an axiom $C \sqsubseteq D_1 \sqcap D_2$, then both D_1 and D_2, as well as all *their* told subsumers, are told subsumers of C. As subsumption is immediate for told subsumers, no tests need to be performed w.r.t. these concepts. In order to maximise the benefit of this optimisation, all of the told subsumers of a concept C are classified *before* C itself is classified.

The told subsumer optimisation can be used to approximate the position of C in the hierarchy: all of its told subsumers, and any super-concepts of these told-subsumers, can be marked as subsumers of C. The most specific concepts in this set of marked concepts are then candidates to be parents of C. In the standard algorithm, however, it is necessary to check (recursively) if the children of these concepts are also subsumers of C. When it has been determined for some subsumer D of C that none of the children of D subsume C, then we know that D is a parent of C.

At the end of the top-down phase we will have computed the set of parents of C; all of the concepts in this set, along with all their super-concepts, are subsumers of C; all other concepts are non-subsumers of C. The next step is to determine the set of children of C (as mentioned above, this step can be omitted for a primitive concept when concepts have been classified in definitional order [2]). This phase is very similar to (the reverse of) the top-down one, and as our optimisation only relates to the top-down phase we won't describe it here—interested readers can refer to [1] for full details.

For large and shallow taxonomies, a concept D may have hundreds or even thousands of children. If, when classifying a concept C, one of its told subsumers is D, then the above algorithm may lead to *all* of the other children of D being

[2] In the FaCT++ implementation we actually use *quasi-definitional order*, as proposed in [5], but to simplify the presentation we will assume that definitional order is used.

checked to see if they subsume C. Although the time taken for each such test may be small, the cumulative cost of all these tests may be prohibitive when classifying such a Tbox. Moreover, in many cases all of these tests will be negative (i.e., no subsumption relationship will be found), and might be thought of as somehow "wasted". The objective of our optimisation is to avoid these "wasted" tests.

3 Completely Defined Concepts

Given a Tbox $\mathcal{T}$, a primitive concept C is said to be completely defined w.r.t. $\mathcal{T}$ when, for the definition $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$ in $\mathcal{T}$, it holds that:

1. For all $1 \leqslant i \leqslant n$, C_i is a primitive concept.
2. (minimality) There exist no $i \neq j$ such that $1 \leq i, j \leq n$ and $C_i \sqsubseteq C_j \sqcap \ldots$.

When the Tbox is obvious from the context we will talk about completely defined concepts without reference to the Tbox.

If we assume a cycle-free Tbox containing only CD concepts and no GCIs, then the classification process is very simple. In fact, we don't need to perform any subsumption tests at all: the position of every concept in the hierarchy is *completely defined* by its told subsumers. If concepts are processed in definitional order, then when a concept C is classified, where C is defined by the axiom $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$, the parents of C are $C_1, \ldots, C_n$, and the only child of C is $\bot$. Note that every concept in such a taxonomy is satisfiable, because there is no use of negation.

The following theorem is straightforward:

Theorem 1 *If a cycle-free KB contains only completely-defined concepts and no general axioms, then the taxonomy built by the above method will be correct.*

This theorem is, however, of very little practical value due to the very stringent conditions on the structure of the Tbox. In the following we will show how the basic technique can be made more useful by weakening some of these conditions.

Primitivity. In general, a CD concept should not have non-primitive concepts in its definition. This is because, when the hierarchy already includes non-primitive concepts (which will be the case given definitional order classification), the bottom-up phase can not be omitted, and the CD method could therefore lead to incorrect results. Assume, e.g., a TBox

$$\{C \sqsubseteq C_1 \sqcap C_2 \sqcap C_3, \quad C' = C_1 \sqcap C_2\}. \tag{1}$$

Using the CD classification approach, C will be classified under C_1, C_2, C_3, whereas it should be classified under C' and C_3.

One case in which this condition can be weakened is for synonyms. A non-primitive concept C is a *synonym* if it's definition is of the form $C = D$, where

D is a primitive concept. Synonyms may came from an application domain, or occur as a result of KB simplification, KB merging, etc.

It is easy to see that synonyms don't require special classification: once D is classified, C will take the same place in the hierarchy. So, adding synonyms to the CD-only KB still allows application of the CD approach.

Minimality. Non-minimal concepts may occur as a result of badly designed ontologies and/or due to absorption of GCIs. The minimality check may, however, be removed from the definition of CD concepts in the classification algorithm. Indeed, checking if each C_i in a definition $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$ is really a parent of C (i.e., has no children that are subsumers of C) is exactly the check that is needed in order to detect non-minimality. This check is relatively cheap and already exists in the classification algorithm.

Non-CD concepts. This is the most important case, because "interesting" ontologies, including most ontologies designed using DL based languages, will contain concept constructors other than conjunction, and this will lead to some concepts being non-CD. This means that, in its current form, the CD approach will usually be largely useless. On the other hand, almost all KBs will contain *some* CD concepts. In this case, it may be possible to split the Tbox into two parts—a CD part (i.e., containing only CD concepts) and a non-CD part—and use the CD algorithm only for the CD part.

Note that such a split will not introduce any problems if the CD part of the classification is performed first—in fact the classification of the CD part is independent of the non-CD part of the Tbox because the definitions of CD concepts only refer to other CD concepts. In the Tbox 1 above, for example, concepts C_1, C_2, C_3 and C will be in the CD part, and C' in the non-CD part. After CD-classification C will have 3 parents, and standard algorithm then insert C' with parents C_1, C_2 and child C.

Cycles. We will distinguish two kinds of definitional cycles. The first (and simplest) is a cycle via concept names, as in the Tbox $K = \{A \sqsubseteq B \sqcap C, B \sqsubseteq A\}$. This kind of cycle can be detected syntactically and transformed into an equivalent definition $K' = \{A \sqsubseteq C, B = A\}$ where A is a CD concept and B is a synonym of A.

Any other kind of terminological cycle must involve non-CD concepts, and so must occur in the non-CD part of TBox. In this case it will be dealt with in the normal way by the standard classification algorithm.

General axioms. GCIs are axioms of the form $C \sqsubseteq D$, where C and D are arbitrary concept expressions.[3] It is easy to see that, in the general case, the CD approach cannot be used in the presence of GCIs. Consider, for example, a Tbox $K = \{C \sqsubseteq \top, \top \sqsubseteq D\}$. In this case, the CD algorithm classifies C under $\top$, whereas it should be classified under D.

[3]Note that, in case there are multiple axioms of the form $C \sqsubseteq D$ or $C \equiv D$ for some concept name C, then only one of these can be considered the definition of C, and the rest must be treated as GCIs (or, in the case of $C \equiv D$, as a pair of GCIs $C \sqsubseteq D$ and $D \sqsubseteq C$).

Fortunately, most realistic KBs contain only general axioms that can be absorbed into either concept implications [6] or role domain restrictions [8], and in this case the CD approach is still applicable.

4 Two-stage Approach Using CD.

The two-stage CD classification algorithm has been implemented in our FaCT++ reasoner as follows. First of all, the following transformations are performed on the Tbox (only transformations relevant to the classification algorithm are mentioned here):

1. Absorb general axioms into concept definitions and/or role domains. If some of the axioms are not absorbable, set `useCD` to `false`. If all the axioms were absorbed, set `useCD` to `true`.

2. Transform simple cycles into sets of synonyms.

3. If `useCD` is `true`, mark some concepts as CD. Namely, $\top$ is marked as CD; a primitive concept C is marked as CD iff it has the definition $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$ and every C_i is marked CD; a non-primitive concept D is marked CD iff it has definition $D = C$ and C is marked CD.

If `useCD` is `true` after the preprocessing, the CD classifier is run prior to the general classifier. The CD classifier works on concepts that are marked CD, processing them in definitional order. For each such concept C, the steps it performs are as follows:

1. If C is a synonym of some already classified concept D, then insert C at the same place as D.

2. For CD C with definition $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$, concepts $C_1, \ldots, C_n$ are candidates to be parents of C.

3. For every candidate C_i, check whether it is redundant, i.e. whether C_i has a child that is an ancestor of a C. This can be done by labelling all ancestors of candidate concepts: labelled candidates will be redundant. Remove redundant candidates from the list of candidates.

4. Insert C into the taxonomy with the remaining candidates as parents and $\bot$ as the only child.

Then the rest of the ontology is then classified using the standard classification algorithm.

We have tested our implementation using several KBs: NCI is the National Cancer Research Institute ontology; GO is the Gene Ontology from the Gene Ontology Consortium; GALEN is the anatomical part of the well-known medical terminology ontology [7]. Details of KB characteristics are given in Table 1,

KB	PConcepts	nCD	NConcepts	Synonyms
NCI	27652	15195	0	0
GO	13926	11718	3	0
GALEN	2048	546	699	18

Table 1: test KB properties

KB	CD	time	nOps	nTests	nCache
NCI	no	76.40	1,614,903	0	10,311,489
	yes	3.61	1,012,281	0	766,054
GO	no	7.40	835,194	30,834	5,184,070
	yes	3.67	783,024	29,768	1,432,892
GALEN	no	204.70	67,524,538	25,660	82,962
	yes	204.54	68,032,698	25,722	43,043

Table 2: test KB results

where PConcepts is the number of primitive concepts, nCD the number of completely defined concepts, NConcepts the number of non-primitive concepts, and Synonyms the number of synonyms. All experiments used v.0.99.4 of FaCT++ running under Linux on an Athlon 1.3GHz machine with 768Mb of memory.

The results of the classification tests are given in Table 2, where time is the time taken to classify the KB (in seconds), nOps is the number of expansion rule applications during the classification process, nTests is the number of subsumption tests, and nCache is the number of subsumptions that were computed using cached models [6].

Using CD speeds up the classification of NCI by a factor of more than 20. In both cases, all subsumption tests are solved cheaply using cached models, but more than ten million tests are performed when CD is not used; employing CD reduces this number to less than one million. Classification of GO is twice as fast with CD than without it. Again, GO has a simple structure, but is very broad, so CD still gives a significant reduction in the large number of cache based tests. GALEN behaves differently. It is the only KB where more "real" (non cache based) subsumption tests are performed with CD than without. This is due to the large number of non-primitive concepts that are classified in the middle of the hierarchy. Even in this case, however, saving large numbers of cache based tests leads to a slightly smaller overall classification time.

5 Discussion

The proposed classification technique is applicable to a large number of real-life ontologies, i.e., those where there are no non-absorbed GCIs. The best results are, of course, for ontologies with large numbers of primitive concepts and a simple structure, but even in cases where it has little beneficial effect, it does not appear to have any detrimental one. The number of such ontologies may decrease, because newly created ontologies will (probably) use more of

the expressive possibilities provided by modern DLs. With legacy ontologies, however, the method may prove to be very useful.

In [5] the so-called *bucket* method was proposed as a way to deal with broad and shallow hierarchies. In this method, when some concept in the hierarchy is found to have a large number of children, a new "virtual" concept is added to the hierarchy; this non-primitive concept is defined to be equivalent to the disjunction of some of the children of the original concept, and is used in fast cache-based comparison.

It is possible to use the bucket method at the second stage of the CD classification algorithm. However, this method will not improve first stage of the algorithm, since no search is actually performed there.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[2] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[3] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-ref/`.

[4] GOA project. European Bioinformatics Institute. `http://www.ebi.ac.uk/GOA/`.

[5] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 161–168, 2001.

[6] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[7] J. E. Rogers, A. Roberts, W. D. Solomon, E. van der Haring, C. J. Wroe, P. E. Zanstra, and A. L. Rector. GALEN ten years on: Tasks and supporting tools. In *Proc. of MEDINFO2001*, pages 256–260, 2001.

[8] D. Tsarkov and I. Horrocks. Efficient reasoning with range and domain constraints. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 41–50, 2004.

Appendix G:

"Ordering heuristics for description logic reasoning" by Dmitry Tsarkov and Ian Horrocks, in Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, Scotland, August 2005. Enclosed with the full electronic version of this report as the file ijcai2005-fppinternals.pdf.

Ordering Heuristics for Description Logic Reasoning

Dmitry Tsarkov and Ian Horrocks
School of Computer Science
University of Manchester
Manchester, UK
traskov|horrocks@cs.man.ac.uk

Abstract

We present a new architecture for Description Logic implementations, a range of new optimisation techniques and an empirical analysis of their effectiveness.

1 Introduction

Description Logics (DLs) are a family of logic based knowledge representation formalisms. Although they have a range of applications (e.g., configuration [McGuinness & Wright, 1998], and reasoning with database schemas and queries [Calvanese *et al.*, 1998b; 1998a]), they are perhaps best known as the basis for widely used ontology languages such as OIL, DAML+OIL and OWL [Horrocks *et al.*, 2003]. As well as DLs providing the formal underpinnings for these languages (i.e., a declarative semantics), DL systems are also used to provide computational services for ontology tools and applications [Knublauch *et al.*, 2004; Rectore, 2003].

Most modern DL systems are based on tableaux algorithms. Such algorithms were first introduced by Schmidt-Schauß and Smolka [Schmidt-Schauß & Smolka, 1991], and subsequently extended to deal with ever more expressive logics [Baader *et al.*, 2003]. Many systems now implement the $\mathcal{SHIQ}$ DL, a tableaux algorithm for which was first presented in [Horrocks *et al.*, 1999]; this logic is very expressive, and corresponds closely to the OWL ontology language. In spite of the high worst case complexity of the satisfiability/subsumption problem for this logic (ExpTime-complete), highly optimised implementations have been shown to work well in many realistic (ontology) applications [Horrocks, 1998].

Optimisation is crucial to the viability of tableaux based systems: in experiments using both artificial test data and application ontologies, (relatively) unoptimised systems performed very badly, often being (at least) several orders of magnitude slower than optimised systems; in many cases, hours of processing time (in some cases even hundreds of hours) proved insufficient for unoptimised systems to solve problems that took only a few milliseconds for an optimised system [Massacci, 1999; Horrocks & Patel-Schneider, 1998]. Modern systems typically employ a wide range of optimisations, including (at least) those described in [Baader *et al.*, 1994; Horrocks & Patel-Schneider, 1999].

Tableaux algorithms try to construct a graph (usually a tree) representation of a model of a concept, the structure of which is determined by syntactic decomposition of the concept. Most implementations employ a space saving optimisation known as the *trace technique* that uses a top-down construction requiring (for PSpace logics) only polynomial space in order to delineate a tree structure that may be exponential in size (with respect to the size of the input concept). For the ExpTime logics implemented in modern systems, however, guaranteeing polynomial space usage is no longer an option. Moreover, for logics that support inverse roles (such as $\mathcal{SHIQ}$), a strictly top down approach is no longer possible as constraints may be propagated both "up" and "down" the edges in the tree.

We describe an alternative architecture for tableaux implementations that uses a (set of) queue(s) instead of (an adaption of) the standard top-down approach. This architecture, which we have implemented in our new FaCT++ system, has a number of advantages when compared to the top-down approach. Firstly, it is applicable to a much wider range of logics, including the expressive logics implemented in modern systems, because it makes no assumptions about the structure of the graph (in particular, whether tree shaped or not), or the order in which the graph will be constructed. Secondly, it allows for the use of more powerful heuristics that try to improve typical case performance by varying the global order in which different syntactic structures are decomposed; in a top-down construction, such heuristics can only operate on a local region of the graph—typically a single vertex.

2 Preliminaries

We present here a brief introduction to DL (in particular $\mathcal{SHIQ}$) syntax, semantics and reasoning; for further details the reader is referred to [Baader *et al.*, 2003].

2.1 Description Logics

Syntax Let $\mathbf{R}$ be a set of *role names* with both transitive and normal role names $\mathbf{R}_+ \cup \mathbf{R}_\mathsf{P} = \mathbf{R}$, where $\mathbf{R}_+ \cap \mathbf{R}_\mathsf{P} = \emptyset$. The set of $\mathcal{SHIQ}$*-roles* (or *roles* for short) is $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$. Let N_C be a set of *concept names*. The set of $\mathcal{SHIQ}$*-concepts* (or *concepts* for short) is the smallest set such that every concept name $C \in N_C$ is a concept, and if C and D are concepts, R is a role, S is a *simple* role[1] and $n \in \mathbb{N}$, then $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, $(\exists R.C)$, $(\leqslant nR.C)$ and $(\geqslant nR.C)$ are also concepts; the last four are called, respectively, value, exists, atmost and atleast restrictions.

For R and S (possibly inverse) roles, $R \sqsubseteq S$ is called a *role inclusion axiom*, and a finite set of role inclusion axioms is called a *role hierarchy*. For C and D (possibly complex)

[1] A simple role is one that is neither transitive nor has any transitive subroles. Restricting number restrictions to simple roles is required for decidability [Horrocks *et al.*, 1999].

concepts, $C \sqsubseteq D$ is called a *general concept inclusion* (GCI), and a finite set of GCIs is called a *TBox*.

Semantics An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ which maps every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for $P \in \mathbf{R}$ and $R \in \mathbf{R}_+$, $\langle x, y\rangle \in P^{\mathcal{I}}$ iff $\langle y, x\rangle \in P^{-\mathcal{I}}$, and if $\langle x, y\rangle \in R^{\mathcal{I}}$ and $\langle y, z\rangle \in R^{\mathcal{I}}$ then $\langle x, z\rangle \in R^{\mathcal{I}}$. The interpretation function $\cdot^{\mathcal{I}}$ of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ maps, additionally, every concept to a subset of $\Delta^{\mathcal{I}}$ such that

$$\begin{aligned}(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, \quad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}},\\ \neg C^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}},\\ (\exists R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x, C) \neq \emptyset\},\\ (\forall R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x, \neg C) = \emptyset\},\\ (\leqslant nR.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \sharp R^{\mathcal{I}}(x, C) \leqslant n\}, \text{ and}\\ (\geqslant nR.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \sharp R^{\mathcal{I}}(x, C) \geqslant n\},\end{aligned}$$

where $\sharp M$ is the cardinality of a set M and $R^{\mathcal{I}}(x, C)$ is defined as $\{y \mid \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$.

An interpretation $\mathcal{I}$ *satisfies* a role hierarchy $\mathcal{R}$ iff $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ for each $R \sqsubseteq S \in \mathcal{R}$, and it satisfies a TBox $\mathcal{T}$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for each $C \sqsubseteq D \in \mathcal{T}$; such an interpretation is called a *model of* $\mathcal{R}$ and $\mathcal{T}$.

A concept C is *satisfiable* w.r.t. a role hierarchy $\mathcal{R}$ and a TBox $\mathcal{T}$ iff there is a model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a *model* of C w.r.t. $\mathcal{R}$ and $\mathcal{T}$. As usual for expressive DLs, subsumption can be reduced to satisfiability, and reasoning w.r.t. a TBox and role hierarchy can be reduced to reasoning w.r.t. a role hierarchy only [Horrocks *et al.*, 1999].

2.2 Tableaux Algorithms

The basic idea behind a tableau algorithm is to take an input concept C and role hierarchy $\mathcal{R}$, and to try to prove the satisfiability of C w.r.t. $\mathcal{R}$ by constructing a model $\mathcal{I}$ of C w.r.t. $\mathcal{R}$. This is done by syntactically decomposing C so as to derive constraints on the structure of such a model. For example, any model of C must, by definition, contain some individual x such that x is an element of $C^{\mathcal{I}}$, and if C is of the form $\exists R.D$, then the model must also contain an individual y such that $\langle x, y\rangle \in R^{\mathcal{I}}$ and y is an element of $D^{\mathcal{I}}$; if D is non-atomic, then continuing with the decomposition of D would lead to additional constraints. The construction fails if the constraints include a *clash* (an obvious contradiction), e.g., if some individual z must be an element of both C and $\neg C$ for some concept C. Algorithms are normally designed so that they are guaranteed to terminate, and guaranteed to construct a model if one exists; such an algorithm is clearly a decision procedure for concept satisfiability.

In practice, algorithms often work on a tree shaped graph that has a close correspondence to a model; this may be because, e.g., models could be non-finite (although obviously finitely representable), or non-trees (although usually tree-like). Typically this will be a labelled graph (usually a tree or collection of trees) where nodes represent individuals in the model, and are labelled with a set of concepts of which they are instances, and edges represent role relationships between pairs of individuals, and are labelled with a set of role names.

The decomposition and construction is usually carried out by applying so called tableaux expansion rules to the concepts in node labels, with one rule being defined for each of the syntactic constructs in the language (with the exception of negation, which is pushed inwards using re-writings such as de Morgan's laws, until it applies only to atomic concepts). For example, the expansion rule for conjunction causes C and D to be added to any node label already containing $C \sqcap D$ (in order to guarantee termination, side conditions prevent rules from being applied if they do not change either the graph or its labelling).

There are two forms of non-determinism in the expansion procedure. In the first place, many rules may be simultaneously applicable, and some order of rule applications must be chosen. From a correctness perspective, this choice is usually irrelevant[2] (because, if there is a model, then it will be found by any expansion ordering), but as we will see later, the order of expansion can have a big effect on efficiency. In the second place, some rules expand the graph non-deterministically; e.g., the expansion rule for disjunction causes *either* C or D to be added to any node label already containing $C \sqcup D$. From a correctness perspective, this choice *is* relevant (because one choice may lead to the successful construction of a model, while another one does not), and is usually dealt with by backtracking search. Although such search must (in the worst case) consider *all* possible expansions, the order in which they are considered can still have a big effect on efficiency.

Two kinds of rule will be of particular interest in the following discussion: *non-deterministic* rules, such as the $\sqcup$-rule mentioned above, and *generating* rules, such as the $\exists$-rule, that add new nodes to the graph. Applying these rules is likely to be more "costly", as they either increase the size of the graph or increase the size of the search space, and they are typically applied with lower priority than other rules.

3 FaCT++ System Architecture

As discussed above, many implementations use a top-down expansion based on the trace technique. The idea of the top-down expansion is to apply the $\exists$-rule with the lowest priority (i.e., only apply this rule when no other rule is applicable); the added refinement of the trace technique is to discard fully expanded sub-trees, so that only a single "trace" (i.e., a branch of the tree) is kept in memory at any one time.

This technique has the advantage of being very simple and easy to implement—a procedure that exhaustively expands a node label can be applied to the current node and then, recursively, to each of its successors. It does, however, have some serious drawbacks. In the first place, for logics with inverse roles, the top-down method simply breaks down as it relies on the fact that rules only ever add concepts to the label of the node to which they are applied or to the label of one of its successor nodes. The result is that, once the rules have been exhaustively applied to a given node label, no further expansion of that label will be possible. In the presence of inverse roles, expansion rules may also add concepts to the labels of predecessor nodes, which could then require further expansion. Moreover, discarding fully expanded sub-trees may no longer be possible, as the expansion of a concept added to the label of a predecessor may cause concepts to be added to the label of a sibling node that had previously been fully expanded.

In the second place, the top down method forces non-deterministic rules to be applied with a higher priority than generating rules. As the size of the search space caused by non-deterministic rule expansions is, in practice, by

[2] Although the correctness of some algorithms requires a priority ordering for different rules.

far the most serious problem for tableaux based systems [Horrocks, 1997], it may be advantageous to apply non-deterministic rules with the lowest priority [Giunchiglia & Sebastiani, 1996]. In fact, top-down implementations typically apply non-deterministic rules with a priority that is lower than that of all of the other rules *except* the generating rules [Horrocks & Patel-Schneider, 1999].

ToDo List Architecture The FaCT++ system was designed with the intention of implementing DLs that include inverse roles, and of investigating new optimisation techniques, including new ordering heuristics. Currently, FaCT++ implements $\mathcal{SHIF}$, a slightly less expressive variant of $\mathcal{SHIQ}$ where the values in atleast and atmost restrictions can only be zero or one.[3]

Instead of the top-down approach, FaCT++ uses a *ToDo list* to control the application of the expansion rules. The basic idea behind this approach is that rules may become applicable whenever a concept is added to a node label. When this happens, a note of the node/concept pair is added to the ToDo list. The ToDo list sorts all entries according to some order, and gives access to the "first" element in the list.

A given tableaux algorithm takes an entry from the ToDo list and processes it according to the expansion rule(s) relevant to the entry (if any). During the expansion process, new concepts may be added to node labels, and hence entries may be added to the ToDo list. The process continues until either a clash occurs or the ToDo list become empty.

In FaCT++ we implement the ToDo list architecture as a set of queues (FIFO buffers). It is possible to set a priority for each rule type (e.g., $\sqcap$ and $\exists$), and a separate queue is implemented for each unique priority. Whenever the expansion algorithm asks for a new entry, it is taken from the non-empty queue with the highest priority, and the algorithm terminates when all the queues are empty. This means that if the $\exists$-rule has a low priority (say 0), and all other rules have the same priority (say 1), then the expansion will be (modulo inverse roles) top-down and breadth first; if stacks (LIFO buffers) were used instead of queues with the same priorities, then the expansion would simulate the standard top-down method.

4 Heuristics

When implementing reasoning algorithms, heuristics can be used to try to find a "good" order in which to apply inference rules (we will call these *rule-ordering* heuristics) and, for non-deterministic rules, the order in which to explore the different expansion choices offered by rule applications (we will call these *expansion-ordering* heuristics). The aim is to choose an order that leads rapidly to the discovery of a model (in case the input is satisfiable) or to a proof that no model exists (in case the input is unsatisfiable). The usual technique is to compute a weighting for each available option, and to choose the option with the highest (or lowest) weight. Much of the "art" in devising useful heuristics is in finding a suitable compromise between the cost of computing the weightings and their accuracy in predicting good orderings.

Such heuristics can be very effective in improving the performance of propositional satisfiability (SAT) reasoners [Freeman, 1995], but finding useful heuristics for description and modal logics has proved to be more difficult. Choosing a good heuristic, or at least not choosing a bad one, is very important: an inappropriate heuristic may not simply fail to improve performance, it may seriously degrade it. Even more problematical is, given a range of possible heuristics, choosing the best one to use for a given (type of) problem.

So far, the heuristics tried with DL reasoners have mainly been adaptions of those already developed for SAT reasoners, such as the well known MOMS heuristic [Freeman, 1995] and Jeroslow and Wang's weighted occurrences heuristic [Jeroslow & Wang, 1990]. These proved to be largely ineffective, and even to degrade performance due to an adverse interaction with backjumping [Baader *et al.*, 2003]. An alternative heuristic, first presented in [Horrocks, 1997], tries to maximise the effect of dependency directed backtracking (backjumping) by preferentially choosing expansions that introduce concept with "old" dependencies. Even this heuristic, however, has relatively little effect on performance with realistic problems, e.g., problems encountered when reasoning with application ontologies.

We conjecture that the standard top-down architecture has contributed to the difficulty in finding useful heuristics as it rules out many possible choices of rule-ordering; in particular, the top-down technique may require generating rules to be applied with a low priority, and so lead to non-deterministic rules being applied before deterministic generating rules. In contrast, the ToDo list architecture gives a much wider range of possible rule orderings, and so has allowed us to investigate a range of new rule-ordering heuristics, in particular heuristics that give non-deterministic rules the lowest priority.

Another factor that has contributed to the weakness of SAT derived heuristics is that they treat concepts as though they were atoms. This is obviously appropriate in the case of propositional satisfiability, but not in the case of concept satisfiability where sub-concepts may have a complex structure. We have also investigated expansion-ordering heuristics that take into account this structure, in particular a concept's size, maximum quantifier depth, and frequency of usage in the knowledge base.

Implementation in FaCT++ The FaCT++ reasoner uses the standard backtracking search technique to explore the different possible expansions offered by non-deterministic rules (such as the $\sqcup$-rule). Before applying a non-deterministic rule, the current state is saved, and when backtracking, the state is restored before re-applying the same rule (with a different expansion choice). When inverse roles are supported, it is possible for a sequence of deterministic rule applications to propagate changes throughout the graph, and it may, therefore, be necessary to save and restore the whole graph structure (in addition to other data structures such as the ToDo list). FaCT++ trys to minimise the potentially high cost of these operations by lazily saving the graph, (i.e., saving parts of the graph only as necessitated by the expansion), but the cost of saving the state still makes it expensive to apply a non-deterministic rule, even if the state is never restored during backtracking.

As discussed in Section 3, FaCT++ uses a ToDo list architecture with separate queues for each priority level. Different rule-ordering heuristics can, therefore, be tried simply by varying the priorities assigned to different rule types. Low priorities are typically given to generating and non-deterministic rules, but the ToDo list architecture allows different priority ordering of these rule types; in contrast, the top-down architecture forces a lower priority to be given to generating rules.

FaCT++ also includes a range of different expansion-ordering heuristics that can be used to choose the order in which to explore the different expansion choices offered by the non-deterministic $\sqcup$-rule. This ordering can be on the ba-

[3] $\mathcal{SHIF}$ corresponds to the OWL-Lite ontology language [Horrocks *et al.*, 2003].

sis of the size, maximum quantifier depth, or frequency of usage of each of the concepts in the disjunction, and the order can be either ascending (smallest size, minimum depth and lowest frequency first) or descending. In order to avoid the cost of repeatedly computing such values, FaCT++ gathers all the relevant statistics for each concept as the knowledge base is loaded, and caches them for later use.

5 Empirical Analysis

In order to evaluate the usefulness of the heuristics implemented in FaCT++, we have carried out an empirical analysis using both real-life ontologies and artificial tests from the DL'98 test suite [Horrocks & Patel-Schneider, 1998].

Ontologies can vary widely in terms of size and complexity (e.g., structure of concepts, and types of axiom used). We used three ontologies with different characteristics in order to see how the heuristics would perform in each case:

WineFood A sample ontology that makes up part of the OWL test suit[4] [Carroll & De Roo, 2004]; it is small, but has a complex structure and includes 150 GCIs.

DOLCE A foundational (top-level) ontology, developed in the WonderWeb project [Gangemi *et al.*, 2002]; it is of medium size and medium complexity.

GALEN The anatomical part of the well-known medical terminology ontology [Rogers *et al.*, 2001]; it is large (4,000 concepts) and has a relatively simple structure, but includes over 400 GCIs.

FaCT++ separates the classification process into satisfiability testing (SAT) and subsumption testing (SUB) phases; the results from the SAT phase are cached and used to speed up subsequent tests via a standard "model-merging" optimisation [Horrocks & Patel-Schneider, 1999]. FaCT++ allows different heuristics to be used in the two phases of the process; this is because the tests have different characteristics: in the SAT phase, nearly all of the tests are satisfiable (ontologies typically do not give names to unsatisfiable concepts), while in the SUB phase, up to one in four of the tests are unsatisfiable. We measured the time (in CPU seconds) taken by FaCT++ to complete each phase.

In addition to the ontologies, we used artificially generated test data from the DL'98 test suite. Artificial tests are in some sense corner cases for a DL reasoner designed primarily for ontology reasoning, and these tests are mainly intended to investigate the effect of hard problems with very artificial structures on the behaviour of our heuristics. For this purpose we selected from the test suite several of the tests that proved to be hard for FaCT++.

Each of these tests consists of a set of 21 satisfiability testing problems of similar structure, but (supposedly exponentially) increasing difficulty; the idea of the test is to determine the number of the largest problem that can be solved within a fixed amount of processing time (100 seconds of CPU time in our case). The names of the tests are of the form "`test_p`" or "`test_n`", where "`test`" refers to the kind of problem (e.g., the "`ph`" tests are derived from encodings of pigeon hole sorting problems), and "`p`/`n`" refers to whether the problems in the test set are satisfiable (`n`) or unsatisfiable (`p`). For these tests we have reported the number of the largest problem solved in less than 100 seconds (21 means that all the problems were solved), along with the time (in CPU seconds) taken for the hardest problem that was successfully solved.

[4] This ontology therefore has a much weaker claim to being "real-life".

For all the tests, FaCT++ v.0.99.2 was used on Pentium 4 2.2 GHz machine with 512Mb of memory, running Linux. Times were averaged over 3 test runs.

5.1 Rule-ordering Heuristics

In these tests we tried a range of different rule-ordering strategies. Each "strategy" is shown as a sequence of letters specifying the priorities (highest first) of the different rule types, where "O" refers to the $\sqcup$-rule, "E" to the $\exists$-rule, and "a" to any other rule type. E.g., "aO" describes the strategy where the $\sqcup$-rule has the lowest priority, and all other rules have an equal higher priority.

Ontology tests The results of using different rule-ordering strategies with the various ontologies are shown in Table 1. All ontologies were tested with the best disjunction-ordering heuristic, as determined in separate tests (see below).

KB	DOLCE		WineFood		GALEN	
	SAT	SUB	SAT	SUB	SAT	SUB
a	0.74	0.74	0.22	2.44	99.44	1678.11
aO	0.64	0.68	**0.14**	**1.64**	29.80	569, 64
aEO	**0.58**	**0.57**	0.15	1.67	**9.88**	**173.79**
aE	0.60	0.58	0.27	2.87	13.35	205.32
aOE	0.61	0.59	0.27	2.93	13.22	201.40

Table 1: Ontology tests with different rule-orderings

The first thing to note is that rule-orderings have relatively little effect on the DOLCE and WineFood ontologies; in contrast, the performance of the best and worst strategies differs by a factor of almost 10 in the GALEN tests. Even in the GALEN case, however, the difference between the "-O" strategies (i.e., those that assign the lowest priority to the $\sqcup$-rule) and "-E" strategies (i.e., those that assign the lowest priority to the $\exists$-rule) is relatively small. In most cases the best result is given by the "aEO" strategy, i.e., by assigning the lowest priority to the $\sqcup$-rule and the next lowest priority to the $\exists$-rule, and even when "aEO" is not the best strategy, the difference between it and the best strategy is very small. Moreover, the difference between the "aEO" and "aOE" strategies is small in most cases, and never more than a factor of 2.

DL98 tests The results of using different rule-ordering strategies with the DL98 tests are shown in Table 2. The first thing to note from these results is that rule-ordering heuristics can have a much more significant effect than in the ontology tests: in some cases the performance of the best and worst strategies differs by a factor of more than 100. In most tests, the "-E" strategies give the best results, with the difference between "-O" and "-E" strategies being much more marked than in the case of the ontology tests. In the case of the `d4_n` test, however, performance is dramatically improved (by a factor of 20) when an "-O" strategy is used.

test	`br_n`		`br_p`		`d4_n`		`ph_n`		`ph_p`	
	last	time	last	time	last	time	last	time	last	time
a	8	16.7	9	20.5	20	94.8	11	99.0	7	15.5
aO	11	38.2	11	38.1	21	0.8	10	10.8	7	32.1
aEO	11	38.8	11	39.0	21	**0.8**	10	10.9	7	32.9
aE	11	**17.1**	12	**18.3**	21	15.7	11	**97.4**	7	**15.2**
aOE	11	19.3	12	21.1	21	16.1	11	99.5	7	15.9

Table 2: DL-98 tests with different rule-ordering strategies

5.2 Expansion-ordering Heuristics

In these tests we tried a range of different expansion-ordering heuristics. Each heuristic is denoted by two letters, the first of

which indicates whether the ordering is based on concept size ("S"), maximum depth ("D") or frequency of usage ("F"), and the second of which indicates ascending ("a") or descending ("d") order. In each group of tests we used the best rule-ordering heuristic as determined by the tests in Section 5.1.

Ontology tests For the ontology tests, we tried different orderings for the SAT and SUB phases of classification. The results are presented in Tables 3, 4 and 5; the first figure in each column is the time taken by the SAT phase using the given ordering, and the remaining figures are the subsequent times taken using different SUB phase orderings.

For DOLCE (Table 3), the difference between the best and worst orderings was a factor of about 4, and many possible orderings were near optimal. For WineFood (Table 4), the difference between the best and worst orderings was a factor of about 2, and using Sd for SAT tests and Dd for SUB tests gave the best result, although several other orderings gave similar results. For GALEN (Table 5), the difference between the best and worst orderings was so large that we were only the orderings given allowed tests to be completed in a reasonable time. The best result was given by using Da for both phases.

SAT	Sa	Da	Fa	Sd	Dd	Fd
SUB	1.29	1.28	1.24	0.61	0.6	0.6
Sa	2.53	2.52	2.52	2.46	2.45	2.41
Da	2.53	2.53	2.53	2.44	2.44	2.41
Fa	0.91	0.91	0.89	0.97	0.98	0.88
Sd	0.61	0.60	0.60	**0.59**	**0.59**	**0.59**
Dd	0.60	0.60	0.60	**0.60**	**0.59**	**0.60**
Fd	1.33	1.34	1.33	1.30	1.34	1.33

Table 3: DOLCE test with different expansion-orderings

SAT	Sa	Da	Fa	Sd	Dd	Fd
SUB	0.26	0.29	0.19	0.13	0.13	0.20
Sa	3.15	3.57	3.27	3.21	3.21	3.68
Da	3.54	3.57	3.44	3.20	3.40	3.47
Fa	3.67	3.57	2.32	2.12	2.41	2.35
Sd	1.77	1.80	1.71	1.80	1.80	1.83
Dd	1.69	1.77	1.87	**1.66**	1.78	1.78
Fd	2.30	2.26	2.75	3.14	3.54	2.76

Table 4: WineFood test with different expansion-orderings

SAT	Sa	Da
SUB	18.76	9.88
Sa	276.90	276.16
Da	185.79	**172.89**
Fd	1049.74	943.06

Table 5: GALEN test with different expansion-orderings

DL98 tests Table 6 presents the results for the DL98 tests. Each column shows the times taken using different expansion orderings to solve the hardest problem that was solvable within the stipulated time limit using any ordering.

In almost every test, the difference between the best and worst strategies is large: a factor of more than 300 in the `d4_n` test. Moreover, strategies that are good in one test can be very bad in another (the Sd and Dd strategies are the best ones in the branch tests (`br_n` and `br_p`), but (by far) the worst in the `d4_n` test), and this is not strongly dependent on the satisfiability result (in the `br` tests, all strategies perform similarly in both satisfiable and unsatisfiable cases). The Fd strategy is, however, either optimal or near optimal in all cases.

order	`br_n`	`br_p`	`d4_n`	`ph_n`	`ph_p`
	test 11	test 12	test 21	test 10	test 7
Sa	22.6	24.8	0.9	8.1	29.5
Da	22.6	24.8	0.9	>300	24.5
Fa	>300	>300	32.0	22.9	20.2
Sd	**17.0**	**18.3**	>300	38.7	24.7
Dd	**17.1**	**18.3**	>300	19.7	19.3
Fd	22.2	25.1	**0.8**	**6.2**	**15.3**

Table 6: DL98 tests with different Or strategies

5.3 Analysis

The different rule-ordering heuristics we tried had relatively little effect on the performance of the reasoner when classifying the DOLCE and WineFood ontologies. With the GALEN ontology, any strategy that gave a lower priority to the $\exists$- and $\sqcup$-rules worked reasonably well, and the aEO strategy was optimal or near-optimal in all cases. The crucial factor with GALEN is giving low priority to the $\exists$-rule. This is due to the fact that GALEN is large, contains many GCIs and also contains existential cycles in concept inclusion axioms (e.g., $C \sqsubseteq \exists R.D$ and $D \sqsubseteq \exists R^{-}.C$); as a result, the graph can grow very large, and this increases both the size of the search space (because GCI related non-determinism may apply on a per-node basis) and the cost of saving and restoring the state during backtracking search. Giving a low priority to the $\exists$-rule minimises the size of the graph and hence can reduce both the size of the search space and the cost of saving and restoring. This effect is less noticeable with the other ontologies because their smaller size and/or lower number of GCIs greatly reduces the maximum size of graphs and/or search space. In view of these results, FaCT++'s default rule-ordering strategy has been set to aEO.[5]

The picture is quite different in the case of the DL'98 tests. Here, different strategies can make a large difference, and no one strategy is universally near optimal. This is to be expected, given that some of the tests include very little non-determinism, but are designed to force the construction of very large models (and hence graphs), while others are highly non-deterministic, but have only very small models. Given that these extreme cases are not representative of typical real-life ontologies, the test results may not be directly relevant to a system designed to deal with such ontologies. It is interesting, however, to see how *badly* the heuristics can behave in such cases: in fact the standard aEO strategy is near optimal in two of the tests, and is never worse than the optimal strategy by a factor of more than 2.

The expansion-ordering heuristics had a much bigger effect on ontology reasoning performance (than the rule-ordering heuristics). In the case of DOLCE and WineFood, almost any strategy that uses Sd or Dd in the SUB phase is near optimal. For GALEN, however, using Da in both phases gives by far the best results. This is again due to the characteristic structure of this ontology, and the fact that preferentially choosing concepts with low modal depth tends to reduce the size of the graph. Unfortunately, no one strategy is universally good (Da/Da is best for GALEN but worst for DOLCE and WineFood); currently, Sd/Dd is the default setting, as the majority of real life ontologies resemble DOLCE and WineFood more than GALEN), but this can of course be changed

[5] Top-down architectures necessarily give lowest priority to the $\exists$-rule, and generally give low priority to $\sqcup$-rule, which is why they work relatively well with ontologies.

by the user if it is known that the ontology to be reasoned with will have a GALEN-like structure.

For the DL'98 tests, the picture is again quite confused: the Sd strategy (the default in the SAT phase) is optimal in some tests, but bad in others—disastrously so in the case of the `d4_n` test. As in the ontology case, the only "solution" offered at present is to allow users to tune these settings according to the problem type or empirical results.

6 Discussion and Future Work

We have described the ToDo list architecture used in the FaCT++ system along with a range of heuristics that can be used for rule and expansion ordering. We have also presented an empirical analysis of these heuristics and shown how these have led us to select the default setting currently used by FaCT++.

These default settings reflect the current predominance of relatively small and simply structured ontologies. This may not, however, be a realistic picture of the kinds of ontology that we can expect in the future: many existing ontologies (including, e.g., WineFood) pre-date the development of OWL, and have been translated from less expressive formalisms. With more widespread use of OWL, and the increasing availability of sophisticated ontology development tools, it may be reasonable to expect the emergence of larger and more complex ontologies. As we have seen in Section 5.1, heuristics can be very effective in helping us to deal efficiently with such ontologies, but choosing a suitable heuristic becomes of critical importance.

In our existing implementation, changing heuristics requires the user to set the appropriate parameters when using the reasoner. This is clearly undesirable at best, and unrealistic for non-expert users. We are, therefore, working on techniques that will allow us to guess the most appropriate heuristics for a given ontology. The idea is to make an initial guess based on an analysis of the syntactic structure of the ontology (it should be quite easy to distinguish GALEN-like ontologies from DOLCE and WineFood-like ontologies simply by examining the statistics that have already been gathered for use in expansion-ordering heuristics), with subsequent adjustments being made based on the behaviour of the algorithm (e.g., the size of graphs being constructed).

Another limitation of the existing implementation is that a single strategy is used for all the tests performed in the classification process. In practice, the characteristics of different tests (e.g., w.r.t. concept size and/or satisfiability) may vary considerable, and it may make sense to dynamically switch heuristics depending on the kind of test being performed. This again depends on having an effective (and cheap) method for analysing the likely characteristics of a given test, and syntactic and behavioural analyses will also be investigated in this context.

References

[Baader *et al.*, 1994] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence*, 4:109–132, 1994.

[Baader *et al.*, 2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds. *The Description Logic Handbook*. CUP, 2003.

[Calvanese *et al.*, 1998a] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of PODS'98*, 1998.

[Calvanese *et al.*, 1998b] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *Proc. of KR'98*, pages 2–13, 1998.

[Carroll & De Roo, 2004] J. Carroll and J. De Roo. OWL web ontology language test cases. W3C Recommendation, 2004.

[Freeman, 1995] J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, 1995.

[Gangemi *et al.*, 2002] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with DOLCE. In *Proc. of EKAW 2002*, 2002.

[Giunchiglia & Sebastiani, 1996] F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for $\mathcal{ALC}$. In *Proc. of KR'96*, pages 304–314, 1996.

[Horrocks *et al.*, 1999] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, pages 161–180, 1999.

[Horrocks *et al.*, 2003] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[Horrocks, 1997] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[Horrocks, 1998] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR'98*, pages 636–647, 1998.

[Horrocks & Patel-Schneider, 1998] I. Horrocks and P. F. Patel-Schneider. DL systems comparison. In *Proc. of DL'98*, pages 55–57, 1998.

[Horrocks & Patel-Schneider, 1999] I. Horrocks and P. F. Patel-Schneider. Optimizing description logic subsumption. *J. of Logic and Computation*, 9(3):267–293, 1999.

[Jeroslow & Wang, 1990] R. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Ann. of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[Knublauch *et al.*, 2004] H. Knublauch, R. Fergerson, N. Noy, and M. Musen. The protégé OWL plugin: An open development environment for semantic web applications. In *Proc. of ISWC 2004*, 2004.

[Massacci, 1999] F. Massacci. TANCS non classical system comparison. In *Proc. of TABLEAUX'99*, 1999.

[McGuinness & Wright, 1998] D. L. McGuinness and J. R. Wright. An industrial strength description logic-based configuration platform. *IEEE Intelligent Systems*, pages 69–77, 1998.

[Rectore, 2003] A. Rectore. Description logics in medical informatics. In *The Description Logic Handbook*, pages 306–346. CUP, 2003.

[Rogers *et al.*, 2001] J. E. Rogers, A. Roberts, W. D. Solomon, E. van der Haring, C. J. Wroe, P. E. Zanstra, and A. L. Rector. GALEN ten years on: Tasks and supporting tools. In *Proc. of MEDINFO2001*, pages 256–260, 2001.

[Schmidt-Schauß & Smolka, 1991] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

Appendix H:

"Optimising Terminological Reasoning for Expressive Description Logics" by Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider, which is being submitted to the Journal of Automated Reasoning. Enclosed with the full electronic version of this report as the file optimised.pdf.

Optimising Terminological Reasoning for Expressive Description Logics

Dmitry Tsarkov, Ian Horrocks and Peter F. Patel-Schneider
Department of Computer Science
University of Manchester, UK, and
Bell Labs Research
Lucent Technologies, USA

April 27, 2006

Abstract. Tableaux algorithms are currently the most widely-used and empirically the fastest algorithms for reasoning in expressive Description Logics, including the important Description Logics $\mathcal{SHIQ}$ and $\mathcal{SHOIQ}$. Achieving a high level of performance on terminological reasoning in expressive Description Logics when using tableaux-based algorithms requires the incorporation of a wide variety of optimisations. The Description Logic system FaCT++ implements a wide variety of such optimisations, some present in other reasoners and some novel or refined in FaCT++. Together these optimisations make FaCT++ one of the fastest current systems for terminological reasoning on expressive Description Logics.

Keywords: Description Logic, Reasoning Systems, Optimisations

1. Introduction

Description Logics (DLs) are a family of logic based knowledge representation formalisms. Although they have a range of applications (e.g., configuration (McGuinness and Wright, 1998) and reasoning with database schemas and queries (Calvanese et al., 1998b; Calvanese et al., 1998a)), they are perhaps best known as the basis for widely used ontology languages such as OIL (Fensel et al., 2001), DAML+OIL (Horrocks et al., 2002) and OWL (Horrocks et al., 2003). As well as DLs providing the formal underpinnings for these ontology languages (by means of the declarative semantics of DLs), DL systems are also used to provide computational services for ontology tools and applications (Knublauch et al., 2004; Rector, 2003). Such services typically include (at least) computing the subsumption hierarchy and checking the consistency of named concepts in an ontology.

Most modern DL systems are based on tableaux algorithms. Such algorithms were first introduced by Schmidt-Schauß and Smolka (Schmidt-Schauß and Smolka, 1991), and subsequently extended to deal with a wide range of different logics (Baader et al., 2003). Many systems now implement the $\mathcal{SHIQ}$ DL, a tableaux algorithm for which was first presented by Horrocks, Sattler, and Tobies (Horrocks et al., 1999), or its extension $\mathcal{SHOIQ}$, whose tableaux algorithm was recently described by Horrocks and Sattler

(Horrocks and Sattler, 2005). These two DLs are very expressive, and correspond closely to the OWL ontology language. (Reasoning in OWL Lite can be reduced to reasoning in $\mathcal{SHIQ}$ and reasoning in OWL DL can be reduced to reasoning in $\mathcal{SHOIQ}$ (Horrocks and Patel-Schneider, 2003).) In spite of the high worst case complexity of the satisfiability/subsumption problem for this logic (ExpTime-complete for $\mathcal{SHIQ}$ and NExpTime-complete for $\mathcal{SHOIQ}$), highly optimised implementations of the tableaux algorithms for $\mathcal{SHIQ}$ and $\mathcal{SHOIQ}$ have been shown to work well in many realistic (ontology) applications (Horrocks, 1998; Pan, 2005; Stevens et al., 2002; Wolstencroft et al., 2005).

Optimisation is crucial to the viability of systems that employ tableaux-based algorithms for DL reasoning: in experiments using both artificial test data and application ontologies, (relatively) unoptimised systems performed very badly, often being (at least) several orders of magnitude slower than optimised systems; in many cases, hours of processing time (in some cases even hundreds of hours) proved insufficient for unoptimised systems to solve problems that took only a few milliseconds for an optimised system (Massacci, 1999; Horrocks and Patel-Schneider, 1998).

The optimisation of tableaux based systems has been the subject of extensive study over the course of the last fifteen years (Baader et al., 1994; Horrocks, 1998; Haarslev and Möller, 2001a; Haarslev and Möller, 2001b; Horrocks, 2003; Horrocks and Sattler, 2002; Tsarkov and Horrocks, 2005b), and modern systems typically employ a wide range of optimisations, including (at least) those described by Baader *et al* (Baader et al., 1994) and Horrocks and Patel-Schneider (Horrocks and Patel-Schneider, 1999).

In this paper we describe the optimisation techniques employed in our FaCT++ reasoner (Tsarkov and Horrocks, 2005b). While focusing mainly on novel techniques and significant refinements and extensions of previously known techniques, we have also described the "standard" techniques as well as some simple techniques that are widely employed but often not reported in the literature. In this way we hope to provide both a report on the novel techniques developed in the FaCT++ system and a reasonably comprehensive survey of the optimisation techniques employed in state of the art DL reasoning systems.

Many of the techniques we will describe could be applied to a wide range of DLs. We will, however, focus on $\mathcal{SHOIQ}$, because

- the expressive power of $\mathcal{SHOIQ}$ subsumes that of most of the DLs discussed in the literature or implemented in DL reasoners;
- $\mathcal{SHOIQ}$ is the logic implemented in our FaCT++ system, and in other state of the art DL reasoners such as Pellet (Sirin et al., 2005b);[1] and

[1] In fact Pellet implements the slightly weaker logic $\mathcal{SHOIN}$.

– as the basis of the W3C OWL web ontology language, $\mathcal{SHOIQ}$ is now very widely used in practice.

DLs usually distinguish between the terminological part of a knowledge base (called the TBox), which describes the structure of the domain of discourse in terms of concepts and roles, and the assertional part (called the ABox), which describes some particular situation in terms of instances of concepts and roles. FaCT++ is primarily designed to support TBox reasoning, as this kind of reasoning is widely used, e.g., in ontology design and maintenance tools such as Protégé (Protégé, 2003) and Swoop (Kalyanpur et al., 2005). However, the expressive power of $\mathcal{SHOIQ}$ is such that it blurs the usual distinction between TBox and ABox, and FaCT++ exploits this to support ABox reasoning. This simple approach would, however, clearly not scale to very large ABoxes, and for a discussion of implementation and optimisation techniques designed to address the problems of reasoning with large ABoxes, the reader is referred to work on optimising ABox reasoning in the Racer system (Haarslev and Möller, 2000; Haarslev and Möller, 2001a).

2. Preliminaries

We present here a brief introduction to description logic syntax and semantics (in particular the syntax and semantics of $\mathcal{SHOIQ}$), and to tableaux algorithms for description logics (in particular for $\mathcal{SHOIQ}$). We will not attempt to give a comprehensive description of the algorithms, or present any proofs; rather we will provide just such details of the basic, unoptimised method as will be necessary in order to understand the subsequent sections on optimisation techniques. The interested reader is referred to other work on tableaux algorithms (such as (Baader and Sattler, 2001)) for further details on tableaux algorithms in general, and the introduction of the $\mathcal{SHOIQ}$ tableaux algorithm (Horrocks and Sattler, 2005) for further details on the $\mathcal{SHOIQ}$ algorithm.

2.1. $\mathcal{SHOIQ}$ Syntax and semantics

$\mathcal{SHOIQ}$ is a very expressive DL that, in addition to the standard Boolean connectives, allows for transitive roles, a hierarchy of sub- and super-roles, inverse (sometimes called converse) roles, qualified cardinality constraints, and nominals (i.e., the ability to refer to individuals in concept expressions). This last feature leads to a blurring of the usual DL distinction between TBox (a set of axioms concerning classes and roles) and ABox (a set of axioms concerning individuals); we will return to this point when discussing the syntax and semantics of $\mathcal{SHOIQ}$ knowledge bases.

DEFINITION 1. *Let* $\mathbf{R}$ *be a set of* role names *with both transitive and normal role names* $\mathbf{R}_+ \cup \mathbf{R}_\mathsf{P} = \mathbf{R}$*, where* $\mathbf{R}_\mathsf{P} \cap \mathbf{R}_+ = \emptyset$*. The set of* $\mathcal{SHOIQ}$-roles *(or* roles *for short) is* $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$*. A* role inclusion axiom *is of the form* $R \sqsubseteq S$*, for two roles* R *and* S*. A* role hierarchy *is a finite set of role inclusion axioms.*

An interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ *consists of a non-empty set* $\Delta^\mathcal{I}$*, the* domain *of* $\mathcal{I}$*, and a function* $\cdot^\mathcal{I}$ *which maps every role to a subset of* $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$ *such that, for* $P \in \mathbf{R}$ *and* $R \in \mathbf{R}_+$*,*

$$\langle x, y\rangle \in P^\mathcal{I} \text{ iff } \langle y, x\rangle \in P^{-\mathcal{I}},$$
$$\text{and if } \langle x, y\rangle \in R^\mathcal{I} \text{ and } \langle y, z\rangle \in R^\mathcal{I}, \text{ then } \langle x, z\rangle \in R^\mathcal{I}.$$

An interpretation $\mathcal{I}$ satisfies a role hierarchy $\mathcal{R}$ *if* $R^\mathcal{I} \subseteq S^\mathcal{I}$ *for each* $R \sqsubseteq S \in \mathcal{R}$*; such an interpretation is called a* model *of* $\mathcal{R}$*.*

We introduce some notation to make the following considerations easier.

1. The inverse relation on roles is symmetric, and to avoid considering roles such as R^{--}, we define a function Inv which returns the inverse of a role: for $R \in \mathbf{R}$, $\mathsf{Inv}(R) := R^-$ and $\mathsf{Inv}(R^-) := R$.

2. Since set inclusion is transitive and $R^\mathcal{I} \subseteq S^\mathcal{I}$ implies $\mathsf{Inv}(R)^\mathcal{I} \subseteq \mathsf{Inv}(S)^\mathcal{I}$, for a role hierarchy $\mathcal{R}$, we introduce $\sqsubseteq^*_\mathcal{R}$ as the transitive-reflexive closure of $\sqsubseteq$ on $\mathcal{R} \cup \{\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$. We use $R \equiv_\mathcal{R} S$ as an abbreviation for $R \sqsubseteq^*_\mathcal{R} S$ and $S \sqsubseteq^*_\mathcal{R} R$.

3. Obviously, a role R is transitive if and only if its inverse $\mathsf{Inv}(R)$ is transitive. However, in cyclic cases such as $R \equiv_\mathcal{R} S$, S is transitive if R or $\mathsf{Inv}(R)$ is a transitive role name. In order to avoid these case distinctions, the function Trans returns true iff R is a transitive role—regardless whether it is a role name, the inverse of a role name, or equivalent to a transitive role name (or its inverse): $\mathsf{Trans}(S, \mathcal{R}) := \mathrm{true}$ if, for some P with $P \equiv S$, $P \in \mathbf{R}_+$ or $\mathsf{Inv}(P) \in \mathbf{R}_+$; $\mathsf{Trans}(S, \mathcal{R}) := \mathrm{false}$ otherwise.

4. A role R is called *simple* w.r.t. $\mathcal{R}$ iff $\mathsf{Trans}(S, \mathcal{R}) = \mathrm{false}$ for all $S \sqsubseteq^*_\mathcal{R} R$.

5. In the following, if $\mathcal{R}$ is clear from the context, then we may abuse our notation and use $\sqsubseteq^*$ and $\mathsf{Trans}(S)$ instead of $\sqsubseteq^*_\mathcal{R}$ and $\mathsf{Trans}(S, \mathcal{R})$.

DEFINITION 2. *Let* N_C *be a set of* concept names *with a subset* $N_I \subseteq N_C$ *of* nominals*. The set of* $\mathcal{SHOIQ}$-concepts *(or* concepts *for short) is the smallest set such that*

1. *every concept name* $C \in N_C$ *is a concept,*

2. *if C and D are concepts and R is a role, then $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, and $(\exists R.C)$ are also concepts (the last two are called universal and existential restrictions, resp.), and*

3. *if C is a concept, R is a simple role[2] and $n \in \mathbb{N}$, then $(\leqslant nR.C)$ and $(\geqslant nR.C)$ are also concepts (called atmost and atleast restrictions, resp.).*

The interpretation function $\cdot^{\mathcal{I}}$ of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ maps, additionally, every concept to a subset of $\Delta^{\mathcal{I}}$ such that

$$\begin{aligned}
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, \quad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \\
\neg C^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \qquad \sharp o^{\mathcal{I}} = 1 \text{ for all } o \in N_I, \\
(\exists R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x, C) \neq \emptyset\}, \\
(\forall R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x, \neg C) = \emptyset\}, \\
(\leqslant nR.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \sharp R^{\mathcal{I}}(x, C) \leqslant n\}, \text{ and} \\
(\geqslant nR.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \sharp R^{\mathcal{I}}(x, C) \geqslant n\},
\end{aligned}$$

where $\sharp M$ is the cardinality of a set M and $R^{\mathcal{I}}(x, C)$ is defined as $\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}$ and $y \in C^{\mathcal{I}}\}$.

DEFINITION 3. *A $\mathcal{SHOIQ}$ knowledge base (KB) is a pair $\langle \mathcal{T}, \mathcal{A}\rangle$, where*

- *$\mathcal{T}$ (the TBox) is a set of* general concept inclusion *(GCI) axioms of the form $C \sqsubseteq D$, where C and D are (possibly complex) concepts, and*

- *$\mathcal{A}$ (the ABox) is a set of axioms of the form $i : C$ and $(i, j) : R$, where C is a (possibly complex) concept, R is a role, and $\{i, j\} \subseteq N_I$ are nominals.*

An interpretation $\mathcal{I}$ satisfies *a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, it satisfies an axiom $i : C$ if $i^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, and it satisfies an axiom $(i, j) : R$ if for some $\langle x, y\rangle \in R^{\mathcal{I}}$, $i^{\mathcal{I}} = \{x\}$ and $j^{\mathcal{I}} = \{y\}$. An interpretation $\mathcal{I}$* satisfies *a TBox $\mathcal{T}$ (resp. an ABox $\mathcal{A}$) if it satisfies each axiom in $\mathcal{T}$ (resp. $\mathcal{A}$), and it satisfies a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ if it satisfies both $\mathcal{T}$ and $\mathcal{A}$; such an interpretation is called a* model of *$\mathcal{T}$ (resp. $\mathcal{A}$, $\mathcal{K}$).*

Note that the use of nominals instead of individuals in ABox axioms leads to the slightly non-standard semantics, but is otherwise insignificant.

As mentioned above, nominals blur the distinction between TBox and ABox. ABox axioms can be transformed into TBox axioms: it is easy to see that an interpretation $\mathcal{I}$ satisfies an axiom $i : C$ iff it satisfies $i \sqsubseteq C$, and it satisfies an axiom $(i, j) : R$ iff it satisfies $i \sqsubseteq \exists R.j$. It is, therefore, possible (and convenient) to think of a $\mathcal{SHOIQ}$ KB as consisting only of a TBox.

[2] Restricting number restrictions to simple roles is required in order to yield a decidable logic (Horrocks et al., 1999).

DEFINITION 4. *A TBox $\mathcal{T}$ is* satisfiable w.r.t. a role hierarchy $\mathcal{R}$ *if there is a model $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{R}$. A concept C is* satisfiable w.r.t. a role hierarchy $\mathcal{R}$ and a TBox $\mathcal{T}$ *if there is a model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a* model of C *w.r.t. $\mathcal{R}$ and $\mathcal{T}$. A concept D* subsumes *a concept C w.r.t. $\mathcal{R}$ and $\mathcal{T}$ (written $C \sqsubseteq_{\mathcal{R},\mathcal{T}} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds in every model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$. Two concepts C, D are* equivalent *w.r.t. $\mathcal{R}$ and $\mathcal{T}$ (written $C \equiv_{\mathcal{R},\mathcal{T}} D$) if they are mutually subsuming w.r.t. $\mathcal{R}$ and $\mathcal{T}$.*

Note that, as usual, subsumption and satisfiability can be reduced to each other: $C \sqsubseteq_{\mathcal{R},\mathcal{T}} D$ if $(C \sqcap \neg D)$ is not satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{T}$; and C is not satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{T}$ if $C \sqsubseteq_{\mathcal{R},\mathcal{T}} \bot$. Moreover, the satisfiability of a concept w.r.t. a role hierarchy and TBox can be reduced to the satisfiability of a TBox w.r.t. a role hierarchy: C is satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{T}$ if $\mathcal{T} \cup \{o \sqsubseteq C\}$ is satisfiable w.r.t. $\mathcal{R}$, where o is a "fresh" nominal, i.e., one that does not occur in $\mathcal{T}$. When $\mathcal{R}$ is obvious from the context (or assumed to be empty) we will talk about TBox satisfiability; when both $\mathcal{R}$ and $\mathcal{T}$ are obvious from the context (or assumed to be empty) we will talk about concept satisfiabiliy and subsumption (written $C \sqsubseteq D$).

2.2. A TABLEAUX ALGORITHM FOR $\mathcal{SHOIQ}$

The basic idea behind the tableaux algorithm for $\mathcal{SHOIQ}$ TBox satisfiability is to take as input a TBox $\mathcal{T}$ and role hierarchy $\mathcal{R}$, and to try to prove the satisfiability of $\mathcal{T}$ w.r.t. $\mathcal{R}$ by demonstrating the existence of a model $\mathcal{I}$ of $\mathcal{T}$ w.r.t. $\mathcal{R}$. This is done by syntactically decomposing $\mathcal{T}$ so as to derive constraints on the structure of such a model. For example, if a nominal o occurs in $\mathcal{T}$, then any model of $\mathcal{T}$ must, by definition, contain some individual x such that $o^{\mathcal{I}} = \{x\}$, and if $o \sqsubseteq \exists R.D$ is an axiom in $\mathcal{T}$, then the model must also contain an individual y such that $\langle x, y\rangle \in R^{\mathcal{I}}$ and y is an element of $D^{\mathcal{I}}$; if D is non-atomic, then continuing with the decomposition of D would lead to additional constraints. The construction fails if the constraints include a *clash* (an obvious contradiction), e.g., if some individual z must be an element of both C and $\neg C$ for some concept C. The algorithm is designed so that it is guaranteed to terminate, and guaranteed to construct a model if one exists; such an algorithm is clearly a decision procedure for $\mathcal{SHOIQ}$ TBox satisfiability.

In practice, the algorithm works on a labelled graph, called a *completion graph*, that has a close correspondence to a model; this is because $\mathcal{SHOIQ}$ models may be non-finite (although obviously finitely representable), and because it is convenient not to explicate edges that may be implied by transitivity (of roles).

For ease of presentation we will, as usual, assume all concepts to be in *negation normal form* (NNF). A concept can be transformed into an equivalent one in NNF by pushing negation inwards, making use of de Morgan's

laws and the duality between existential and universal restrictions, and between atmost and atleast number restrictions (Horrocks et al., 2000). For a concept C, we use $\doteq\neg C$ to denote the NNF of $\neg C$, and we use $\mathsf{cl}(C)$ to denote the set of all subconcepts of C (including C). We will also abuse our notation by saying that a TBox $\mathcal{T}$ is in NNF if all the concepts occurring in $\mathcal{T}$ are in NNF, and by using $\mathsf{cl}(\mathcal{T})$ to denote the set of all subconcepts of the concepts occurring in $\mathcal{T}$, i.e., $\bigcup_{(C\sqsubseteq D)\in\mathcal{T}} \mathsf{cl}(C) \cup \mathsf{cl}(D)$.

DEFINITION 5. *Let $\mathcal{R}$ be a role hierarchy and $\mathcal{T}$ a $\mathcal{SHOIQ}$ TBox in NNF. A* completion graph *for $\mathcal{T}$ with respect to $\mathcal{R}$ is a directed graph $\mathbf{G} = (V, E, \mathcal{L}, \not\doteq)$ where each node $x \in V$ is labelled with a set $\mathcal{L}(x) \subseteq \mathsf{cl}(\mathcal{T})$ and each edge $\langle x, y\rangle \in E$ is labelled with a set of role names $\mathcal{L}(\langle x, y\rangle)$ containing (possibly inverse) roles occurring in $\mathcal{T}$ or $\mathcal{R}$. Additionally, we keep track of inequalities between nodes of the graph with a symmetric binary relation $\not\doteq$ between the nodes of $\mathbf{G}$.*

If $\langle x, y\rangle \in E$, then y is called a successor *of x and x is called a* predecessor *of y.* Ancestor *is the transitive closure of predecessor, and* descendant *is the transitive closure of successor. A node y is called an R-successor of a node x if, for some R' with $R' \sqsubseteq^* R$, $R' \in \mathcal{L}(\langle x, y\rangle)$; x is called an R-predecessor of y if y is an R-successor of x. A node y is called a* neighbour *(R-*neighbour*) of a node x if y is a successor (R-successor) of x or if x is a successor (R^--successor) of y.*

For a role S and a node x in $\mathbf{G}$, we define the set of x's S-neighbours with C in their label, $S^{\mathbf{G}}(x, C)$, as follows:

$$S^{\mathbf{G}}(x, C) := \{y \mid y \text{ is an } S\text{-neighbour of } x \text{ and } C \in \mathcal{L}(y)\}.$$

$\mathbf{G}$ is said to contain a clash *if*

1. *for some $A \in N_C$ and node x of $\mathbf{G}$, $\{A, \neg A\} \subseteq \mathcal{L}(x)$,*
2. *for some node x of $\mathbf{G}$, $(\leqslant nS.C) \in \mathcal{L}(x)$ and there are $n+1$ S-neighbours $y_0, \ldots, y_n$ of x with $C \in \mathcal{L}(y_i)$ for each $0 \leq i \leq n$ and $y_i \not\doteq y_j$ for each $0 \leq i < j \leq n$,* or
3. *for some $o \in N_I$, there are two nodes x, y of $\mathbf{G}$ with $x \not\doteq y$ and $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$.*

If $o_1, \ldots, o_\ell$ are all the nominals occurring in $\mathcal{T}$, then the tableau algorithm starts with the completion graph $\mathbf{G} = (\{r_0, r_1 \ldots, r_\ell\}, \emptyset, \mathcal{L}, \emptyset)$ where $\mathcal{L}(r_0) = \{D\}$ and $\mathcal{L}(r_i) = \{o_i\}$ for $1 \leq i \leq \ell$. $\mathbf{G}$ is then expanded by repeatedly applying the expansion rules given in Figures 1 and 2, stopping if a clash occurs.

The expansion rules use some auxiliary terms and operations, defined here:

Nominal Nodes and Blockable Nodes We distinguish two types of nodes in $\mathbf{G}$, *nominal* nodes and *blockable* nodes. A node x is a nominal node if $\mathcal{L}(x)$ contains a nominal. A node that is not a nominal node is a blockable node. A nominal $o \in N_I$ is said to be *new in* $\mathbf{G}$ if no node in $\mathbf{G}$ has o in its label.

Blocking A node x is *label blocked* if it has ancestors x', y and y' such that

1. x is a successor of x' and y is a successor of y',
2. y, x and all nodes on the path from y to x are blockable,
3. $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x') = \mathcal{L}(y')$, *and*
4. $\mathcal{L}(\langle x', x\rangle) = \mathcal{L}(\langle y', y\rangle)$.

In this case, we say that y *blocks* x. A node is *blocked* if either it is label blocked or it is blockable and its predecessor is blocked; if the predecessor of a blockable node x is blocked, then we say that x is *indirectly blocked*.

Generating and Shrinking Rules and Safe Neighbours The rules $\geqslant$-rule, $\exists$-rule, and $\mathsf{o?}$-rule are called *generating rules*, and the rules $\leqslant$-rule and o-rule are called *shrinking rules*. An R-neighbour y of a node x is *safe* if (i) x is blockable or if (ii) x is a nominal node and y is not blocked.

Pruning When a node y is *merged* into a node x, we "prune" the completion graph by removing y and, recursively, all blockable successors of y. More precisely, pruning a node y (written $\mathsf{Prune}(y)$) in $\mathbf{G} = (V, E, \mathcal{L}, \not\doteq)$ yields a graph that is obtained from $\mathbf{G}$ as follows:

1. for all successors z of y, remove $\langle y, z\rangle$ from E and, if z is blockable, $\mathsf{Prune}(z)$;
2. remove y from V.

Merging merging a node y into a node x (written $\mathsf{Merge}(y, x)$) in $\mathbf{G} = (V, E, \mathcal{L}, \not\doteq)$ yields a graph that is obtained from $\mathbf{G}$ as follows:

1. for all nodes z such that $\langle z, y\rangle \in E$
   a) if $\{\langle x, z\rangle, \langle z, x\rangle\} \cap E = \emptyset$, then add $\langle z, x\rangle$ to E and set $\mathcal{L}(\langle z, x\rangle) = \mathcal{L}(\langle z, y\rangle)$,
   b) if $\langle z, x\rangle \in E$, then set $\mathcal{L}(\langle z, x\rangle) = \mathcal{L}(\langle z, x\rangle) \cup \mathcal{L}(\langle z, y\rangle)$,
   c) if $\langle x, z\rangle \in E$, then set $\mathcal{L}(\langle x, z\rangle) = \mathcal{L}(\langle x, z\rangle) \cup \{\mathsf{Inv}(S) \mid S \in \mathcal{L}(\langle z, y\rangle)\}$, and
   d) remove $\langle z, y\rangle$ from E;

$\sqcap$-rule: if 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and
2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$

$\sqcup$-rule: if 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and
2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

$\exists$-rule: if 1. $\exists S.C \in \mathcal{L}(x)$, x is not blocked, and
2. x has no safe S-neighbour y with $C \in \mathcal{L}(y)$,
then create a new node y with $\mathcal{L}(\langle x, y\rangle) = \{S\}$
and $\mathcal{L}(y) = \{C\}$

$\forall$-rule: if 1. $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked, and
2. there is an S-neighbour y of x with $C \notin \mathcal{L}(y)$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

$\forall_+$-rule: if 1. $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked, and
2. there is some R with $\mathsf{Trans}(R)$ and $R \sqsubseteq^* S$,
3. there is an R-neighbour y of x with $\forall R.C \notin \mathcal{L}(y)$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$

?-rule: if 1. $(\leqslant n S.C) \in \mathcal{L}(x)$, x is not indirectly blocked, and there
2. is an S-neighbour y of x with $\{C, \doteq\neg C\} \cap \mathcal{L}(y) = \emptyset$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{E\}$ for some $E \in \{C, \doteq\neg C\}$

$\geqslant$-rule: if 1. $(\geqslant n S.C) \in \mathcal{L}(x)$, x is not blocked, and
2. there are not n safe S-neighbours $y_1, \ldots, y_n$ of x with
$C \in \mathcal{L}(y_i)$ and $y_i \neq\!\!\!\doteq y_j$ for $1 \leq i < j \leq n$
then create n new nodes $y_1, \ldots, y_n$ with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$,
$\mathcal{L}(y_i) = \{C\}$, and $y_i \neq\!\!\!\doteq y_j$ for $1 \leq i < j \leq n$.

$\leqslant$-rule: if 1. $(\leqslant n S.C) \in \mathcal{L}(z)$, z is not indirectly blocked, and
2. $\sharp S^{\mathbf{G}}(z, C) > n$ and there are two S-neighbours
x, y of z with $C \in \mathcal{L}(x) \cap \mathcal{L}(y)$, and not $x \neq\!\!\!\doteq y$
then 1. if x is a nominal node, then $\mathsf{Merge}(y, x)$
2. else if y is a nominal node or an ancestor of x,
then $\mathsf{Merge}(x, y)$
3. else $\mathsf{Merge}(y, x)$

$\sqsubseteq$-rule: if 1. $C_1 \sqsubseteq C_2 \in \mathcal{T}$, x is not indirectly blocked, and
2. $\{\doteq\neg C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{\doteq\neg C_1, C_2\}$

Figure 1. Basic tableaux expansion rules for $\mathcal{SHOIQ}$

2. for all nominal nodes z such that $\langle y, z\rangle \in E$
   a) if $\{\langle x, z\rangle, \langle z, x\rangle\} \cap E = \emptyset$, then add $\langle x, z\rangle$ to E and set $\mathcal{L}(\langle x, z\rangle) = \mathcal{L}(\langle y, z\rangle)$,
   b) if $\langle x, z\rangle \in E$, then set $\mathcal{L}(\langle x, z\rangle) = \mathcal{L}(\langle x, z\rangle) \cup \mathcal{L}(\langle y, z\rangle)$,
   c) if $\langle z, x\rangle \in E$, then set $\mathcal{L}(\langle z, x\rangle) = \mathcal{L}(\langle z, x\rangle) \cup \{\mathsf{Inv}(S) \mid S \in \mathcal{L}(\langle y, z\rangle)\}$, and
   d) remove $\langle y, z\rangle$ from E;
3. set $\mathcal{L}(x) = \mathcal{L}(x) \cup \mathcal{L}(y)$;
4. add $x \not\doteq z$ for all z such that $y \not\doteq z$; and
5. $\mathsf{Prune}(y)$.

Two kinds of rule will be of particular interest in the following discussion: *non-deterministic* rules, such as the $\sqcup$-rule and $\sqsubseteq$-rule, and *generating* rules, such as the $\exists$-rule and $\geqslant$-rule. In practice, non-deterministic rules are dealt with by using backtracking search to investigate the completion graphs resulting from each possible expansion. Applying such rules is, therefore, likely to be more "costly", as they either increase the size of the graph or increase the size of the search space.

o-rule:	if	for some $o \in N_I$ there are 2 nodes x, y with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ and not $x \not\doteq y$
	then	$\mathsf{Merge}(x, y)$
o?-rule:	if	1. $(\leqslant nS.C) \in \mathcal{L}(x)$, x is a nominal node, and there is a blockable S-neighbour y of x such that $C \in \mathcal{L}(y)$ and x is a successor of y 2. there is no m with $1 \leqslant m \leqslant n$, $(\leqslant mS.C) \in \mathcal{L}(x)$, and there are m nominal S-neighbours $z_1, \ldots, z_m$ of x with $C \in \mathcal{L}(z_i)$ and $y_i \not\doteq y_j$ for all $1 \leq i < j \leq m$.
	then	1. guess $m \leqslant n$ and set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{(\leqslant mS.C)\}$ 2. create m new nodes $y_1, \ldots, y_m$ with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$, $\mathcal{L}(y_i) = \{C, o_i\}$ for $o_i \in N_I$ new in $\mathbf{G}$, and $y_i \not\doteq y_j$ for $1 \leq i < j \leq m$,

Figure 2. Expansion rules dealing with nominals

3. Preprocessing and simplifications

The first group of optimisations in FaCT++ are performed directly on the syntax of the input KB. These optimisations serve to preprocess and simplify the KB into a form more amenable to later processing. As well as simplifications such as tautology elimination and obvious clash detection, preprocessing optimisations (like absorption, Section 3.2.5) can also lead to significant simplification of the subsequent reasoning process.

Note that satisfiability checking for expressive DLs requires, in the worst case, time that is at least exponential in the size of the input, whereas most preprocessing optimisations have polynomial, or even linear worst case complexity.

3.1. LEXICAL NORMALISATION AND SIMPLIFICATION

Descriptions of DL tableau algorithms, such as the one given in Section 2, typically assume that the input is in negation normal form (NNF); this simplifies the (description of the) algorithm, but it means that the first (and most common) kind of clash, i.e., $\{C, \neg C\} \subseteq \mathcal{L}(x)$ for some node x in the completion graph, will only be detected when C is a named concept. For example, when testing the satisfiability of the concept $(A \sqcap B) \sqcap \neg(A \sqcap B)$, the transformation into NNF would give $(A \sqcap B) \sqcap (\neg A \sqcup \neg B)$; in practice this means that, in spite of the "obvious" contradiction, backtracking search will be performed in order to determine that the concept is unsatisfiable.

For this reason, practical algorithms do not transform the input into NNF, but include a $\neg$-rule that performs a single (negation) normalisation step (e.g., applying the $\neg$-rule to $\neg(A \sqcap B) \in \mathcal{L}(x)$ would cause $\neg A \sqcup \neg B$ to be added to $\mathcal{L}(x)$), and the completion graph is said to contain a clash if it contains a node x with $\{C, \neg C\} \subseteq \mathcal{L}(x)$ for an arbitrary concept C. Moreover, in order to facilitate the detection of such clashes, the input is *normalised* and *simplified* so that logically equivalent concepts are more often syntactically equivalent. This is achieved by (recursively) applying a set of rewrite rules to concepts expressions, and by ordering conjuncts w.r.t. some total ordering. For example, we re-write $\sqcup$ and $\exists$ concepts as negated $\sqcap$ and $\forall$ concepts, respectively; we merge several conjunctions together; we order conjuncts; and we use logical equivalences and semantics preserving transformations in order to simplify concepts. These equivalences and transformations can be split into three groups: *constant elimination*, *syntactic equivalences* and *semantic transformations*.

The constant elimination group includes the following equivalences:

$(C \sqcap \top) \equiv C$, $(C \sqcap \bot) \equiv \bot$, $\forall R.\top \equiv \top$.

The syntactic equivalence group includes the following equivalences:

$(C \sqcap C) \equiv C$, $\neg\neg C \equiv C$, $C \sqcap \neg C \equiv \bot$, $\geq 1R.C \equiv \exists R.C$

The semantic transformation group exploits semantic information already gathered during the preprocessing phase to simplify concept expressions. For example, if the KB contains an axiom $A \sqsubseteq B$ for named concepts A and B, then the concept expression $A \sqcap B$ can be simplified to A. The same is true for arbitrary concept expressions.

Rewrite rules can be separated into two classes: "cheap" ones and "expensive" ones. Cheap ones (i.e., $C \sqcap \bot \rightarrow \bot$) are easily applied and almost always give a positive effect. They are, therefore, applied to every concept expression appearing in the KB.

Expensive rules require more effort to recognise when they are applicable (i.e., $A \sqcap B \rightarrow A$ when $A \sqsubseteq B$), and may not give a positive effect at all. Indeed, a well designed KB is likely to include few such constructions, so applying such rule to the whole (possibly huge) KB will be a waste of time. However, there are special cases in which it is crucial to have concept expression be as simple as possible. Such cases include unabsorbed axioms, absorbed concept expressions and role ranges and domains (all of which will be explained below). In these cases, all simplification rules (including the expensive ones) are applied.

3.2. DEALING WITH AXIOMS

If dealt with naively, TBox axioms can lead to a serious degradation in reasoning performance, as each such axiom would cause a disjunction to be added to every node of the completion graph, leading to potentially enormous amounts on nondeterministic expansion and backtracking search.

For example, given a TBox $\mathcal{T}$, if $\top \sqsubseteq A \in \mathcal{T}$, with $A = ((C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n))$, and testing the satisfiability of $\mathcal{T}$ leads to the construction of a completion graph containing k nodes, then there are 2^{kn} different ways to apply the $\sqcap$- and $\sqcup$-rules to the resulting k copies of A. This explosion in the size of the search space can easily lead to a catastrophic degradation in performance, even when optimisations such as backjumping (see Section 4.3) and caching (see Section 4.6) are employed (Horrocks, 1997).

Fortunately, optimisations known as *lazy unfolding* and *absorption* have proved to be very effective in reducing the size of the search space.

3.2.1. *Lazy Unfolding*

TBox axioms are often (restricted to be) of the form $A \sqsubseteq C$ or $A \equiv C$ for some concept name A (where $A \equiv C$ is an abbreviation for the pair of GCIs, $A \sqsubseteq C$ and $\neg A \sqsubseteq \neg C$). Such axioms are often called *definitional*, as they can be thought of as defining the meaning of A. A TBox

$$\mathcal{T} = \{A_1 \equiv C_1, \ldots, A_\ell \equiv C_\ell, A_{\ell+1} \sqsubseteq C_{\ell+1}, \ldots, A_{\ell+m} \sqsubseteq C_{\ell+m}\}$$

is said to be *unfoldable*, if it satisfies the following conditions.

– All axioms in $\mathcal{T}$ are definitional.

– Axioms in $\mathcal{T}$ are *unique*. I.e., for each concept name A, $\mathcal{T}$ contains at most one axiom of the form $A \equiv C$ (i.e., $A_i \neq A_j$ for $1 \leqslant i < j \leqslant \ell$), and if it contains an axiom of the form $A \equiv C$, then it does not contain any axiom of the form $A \sqsubseteq C$. (Note that an arbitrary set of axioms $\{A \sqsubseteq C_1, \ldots, A \sqsubseteq C_n\}$ can be combined into a single axiom $A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$).

– $\mathcal{T}$ is *acyclic*. I.e., there is no axiom $A_i \equiv C_i \in \mathcal{T}$ such that A_i occurs either directly or indirectly in C_i.[3] A concept name A occurs indirectly in a concept expression C if there is a concept name A' such that A' occurs directly in C, and there is an axiom $A' \equiv C' \in \mathcal{T}$ such that A occurs either directly or indirectly in C'.

Instead of being dealt with using the $\sqsubseteq$-rule, such a set of axioms can be lazily *unfolded* during the tableau expansion. I.e., for an axiom $A_1 \sqsubseteq C_1 \in \mathcal{T}$, if A_i is added to $\mathcal{L}(x)$ for some node x, then C_i is also added to $\mathcal{L}(x)$, and for an axiom $A_j \equiv C_j \in \mathcal{T}$, if A_j ($\neg A_j$) is added to $\mathcal{L}(x)$ for some node x, then C_j (resp. $\neg C_j$) is also added to $\mathcal{L}(x)$.

It is obvious that an arbitrary TBox $\mathcal{T}$ can be divided into an unfoldable part $\mathcal{T}_u$ and a general part $\mathcal{T}_g$ such that $\mathcal{T}_u \cup \mathcal{T}_g = \mathcal{T}$ and $\mathcal{T}_u \cap \mathcal{T}_g = \emptyset$. The unfoldable part $\mathcal{T}_u$ can then be dealt with using lazy unfolding while the general part $\mathcal{T}_g$ is dealt with using the $\sqsubseteq$-rule. In fact it has been shown that the definition of an unfoldable TBox can be extended somewhat while still allowing the use of the above lazy unfolding technique. In particular, the concept expressions occurring on the left hand side of subsumptions and equivalences can also be negated named concepts, and the acyclicity condition can be relaxed by distinguishing positive and negative occurrences of named concepts in a stratified theory (Horrocks and Tobies, 2000b; Lutz, 1999).

Lazy unfolding can be viewed as a modification of the tableaux expansion rules, replacing the $\sqsubseteq$-rule with two rules, one for unfoldable axioms and one for general axioms. These two rules, the $\sqsubseteq_u$-rule and the $\sqsubseteq_g$-rule are given in Figure 3.

A form of lazy unfolding can also be used to deal more efficiently with so called *range* and *domain* constraints. These often arise in KBs derived from ontologies, where it is common to state, e.g., that the role "drives" has domain "adult" and range "vehicle", where adult and vehicle are concepts, and where the intuitive meaning is that only adults can drive, and that only vehicles can be driven. This can easily be expressed as $\mathcal{SHOIQ}$ axioms of the form

[3] For the purposes of lazy unfolding, only cycles consisting entirely of $\equiv$ axioms are problematical.

$\sqsubseteq_u$-rule: if 1. $C_1 \in \mathcal{L}(x)$, $C_1 \sqsubseteq C_2 \in \mathcal{T}_u$, x is not indirectly blocked, and
2. $\{C_2\} \cap \mathcal{L}(x) = \emptyset$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2\}$

$\sqsubseteq_g$-rule: if 1. $C_1 \sqsubseteq C_2 \in \mathcal{T}_g$, x is not indirectly blocked, and
2. $\{\dot{\neg}C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{\dot{\neg}C_1, C_2\}$

Figure 3. Tableaux expansion rules for unfoldable and general axioms

R-rule: if 1. $\top \sqsubseteq \forall S.C \in \mathcal{T}_r$, x is not indirectly blocked, and
2. there is an S-neighbour y of x with $C \notin \mathcal{L}(y)$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{C\}$

Figure 4. Tableaux expansion rule for domain and range axioms

$\top \sqsubseteq \forall R^{-}.C$ (to state that the domain of R is C, e.g., $\top \sqsubseteq \forall \text{drives}^{-}.\text{adult}$), and $\top \sqsubseteq \forall R.C$ (to state that the range of R is C, e.g., $\top \sqsubseteq \forall \text{drives.vehicle}$).

Such axioms are not unfoldable, and will therefore be dealt with by the $\sqsubseteq$-rule. For an axiom $\top \sqsubseteq \forall R.C$, this would lead to $\neg\top \sqcup \forall R.C$ being added to the label of each node. This can clearly be simplified to $\forall R.C$, so there is no need for non-deterministic expansion. Even so, when there are very large numbers of range and domain axioms (which is the case in some KBs derived from ontologies, where it is common practice to specify the range and domain of *every* role), this may lead to a significant degradation of performance simply due to the large size of node labels (and using complex data structures for node labels is also problematical due to the saving and restoring that is needed during backtracking search). Extending the tableaux algorithm to lazily unfold range and domain axioms not only deals with this problem, but also provides additional opportunities for applying the important *absorption* optimisation (see Section 3.2.5).

It is obvious that an arbitrary TBox $\mathcal{T}$ can be divided into three parts: an unfoldable part $\mathcal{T}_u$, a range and domain part $\mathcal{T}_r$, and a general part $\mathcal{T}_g$, such that $\mathcal{T} = \mathcal{T}_u \cup \mathcal{T}_r \cup \mathcal{T}_g$, and $\mathcal{T}_u$, $\mathcal{T}_r$ and $\mathcal{T}_g$ are pairwise disjoint. The unfoldable part $\mathcal{T}_u$ is as before, the range and domain part $\mathcal{T}_r$ consists of all the axioms of the form $\top \sqsubseteq \forall R.C$ that would formerly have been in $\mathcal{T}_g$ (note that R can be an inverse role), and $\mathcal{T}_g$ consists of the remaining general axioms. A new tableaux expansion rule, the **R**-rule, is added in order to deal with $\mathcal{T}_r$; this rule is given in Figure 4.

3.2.2. *Synonym replacement*
Assume that the only axiom with named concept A on the left hand side is $A \equiv B$, where B is also a named concept. This means that there exist two names for the same concept. This redundancy in the TBox may prevent the application of further optimisations, and may introduce "fake" cycles and dependencies into the TBox. To avoid this, we apply *synonym replacement*.

If $A \equiv B \in \mathcal{T}_u$, then A is called a *synonym* of B, B is called the *representative concept* of A, and the axiom $A \equiv B$ is called the *synonym definition* of A. During synonym replacement, all synonyms occurring in the TBox (except in synonym definitions) are replaced with their representative concepts. After this, other simplifications, such as those described in Section 3.1, can be applied. For example, the concept expression $A \sqcap \neg B$, where A is synonym of B, would be transformed to $B \sqcap \neg B$, which can be further simplified to $\bot$.

3.2.3. *Told Cycle Elimination*
The idea of a told subsumer is widely used in DL reasoning, especially in classification algorithms (see Section 5 below). Informally, a named concept C has a *told subsumer* D if $C \sqsubseteq D$ is "obvious" from the syntactic structure of the KB. This information is useful in classification, because it provides some initial information about subsumption relations.

More formally, a concept D *appears in the expression* C, if either $C = D$, or $C = C_1 \sqcap \ldots \sqcap C_n$ and D appears in the expression C_i for some $1 \leq i \leq n$. A named concept B is called an *immediate told subsumer* (ITS) of a named concept A, written $B \in ITS(A)$, iff:

- $A \sqsubseteq C \in \mathcal{T}$ or $A \equiv C \in \mathcal{T}$ and B appears in the expression C;
- $A \sqsubseteq C \in \mathcal{T}$ or $A \equiv C \in \mathcal{T}$, $\exists R.D$ appears in the expression C, $\top \sqsubseteq \forall\, \mathsf{Inv}(R).E \in \mathcal{T}_r$ and B appears in the expression E;
- $A \sqsubseteq C \in \mathcal{T}$ or $A \equiv C \in \mathcal{T}$, $\geq nR.D$ appears in the expression C, $\top \sqsubseteq \forall\, \mathsf{Inv}(R).E \in \mathcal{T}_r$ and B appears in the expression E.

Told subsumer is the transitive closure of ITS.

A TBox $\mathcal{T}$ has a *definitional cycle* if, for some $A \in N_C$, A is a told subsumer for itself. The presence of definitional cycles in the TBox can lead to several problems, and in particular can cause problems for algorithms that exploit the told subsumer hierarchy (i.e., the concept hierarchy that is implied by told subsumer relations). Definitional cycles also make some GCI simplifications inapplicable. These cycles are, however, often quite easy to eliminate. We assume that most modern reasoners include such an optimisation, although we know of no reference to it in the literature.

Assume $A_1 \ldots A_n$ are named concepts, such that $A_{i+1} \in ITS(A_i)$ for all $1 \leq i < n$, and $A_1 = A_n$. In this case, all A_i are equivalent. We can chose A_1

as a representative concept, make $A_2, \ldots, A_{n-1}$ synonyms of A_1 by adding axioms $A_i \equiv A_1$ to $\mathcal{T}_u$, and run the synonym replacement algorithm on $\mathcal{T}$.

3.2.4. *Redundant subsumption elimination*

After applying told cycle elimination, some axioms might include redundant information due to synonym replacement. For example, if $A \sqsubseteq B \sqcap C \in \mathcal{T}$, and after told cycle elimination $B \equiv A$ was added to $\mathcal{T}_u$, then the axiom would be transformed into $A \sqsubseteq A \sqcap C$, where the A in $A \sqcap C$ is clearly redundant. *Redundant subsumption elimination* is a simple optimisation that removes this kind of redundancy.

The optimisation works as follows. For every axiom $A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n \in \mathcal{T}_u$, and for $1 \leqslant i \leqslant n$, if $C_i = A$, then C_i is replaced with $\top$; the usual syntactic simplifications are then applied to the axiom.

Note that this optimisation is different from told cycle elimination (Section 3.2.3), although there are some axioms that can be simplified using either optimisation; the axiom $A \sqsubseteq A \sqcap \neg B$ would, for example, be simplified to $A \sqsubseteq \neg B$ by either optimisation. Told cycle elimination cannot, however, deal with axioms like $A \sqsubseteq \neg A \sqcup C$, and redundant subsumption elimination is not applicable if several axioms are involved in a cycle.

3.2.5. *Absorption*

Given the effectiveness of lazy unfolding in dealing with the unfoldable part of a TBox $\mathcal{T}_u$ and the range and domain axioms in $\mathcal{T}_r$, it makes sense to try to rewrite the axioms in $\mathcal{T}$ so that the size of $\mathcal{T}_g$ can be reduced.

Absorption is such a rewriting optimisation that tries to eliminate GCIs in $\mathcal{T}_g$ by absorbing them into concept definitions in $\mathcal{T}_u$ (*concept absorption*) or domain axioms in $\mathcal{T}_r$ (*role absorption*). This is usually done by rewriting generals axiom in an equivalent form suitable for one of these absorptions: for concept absorption, the axiom should be of the form $CN \sqsubseteq D$, where CN is a named primitive concept and D is an arbitrary concept expression (Horrocks, 2003); for role absorption, the axiom should be of the form $\exists R.\top \sqsubseteq D$, where D is an arbitrary concept expression (Tsarkov and Horrocks, 2004). In addition, a special form of concept absorption, called nominal absorption, can be employed when an axiom has form $o_1 \sqcup \ldots \sqcup o_n \sqsubseteq D$, or $\exists R.o \sqsubseteq D$, where $o, o_1, \ldots, o_n$ are nominals and D is an arbitrary concept expression (Sirin et al., 2005a).

Given a TBox $\mathcal{T}$, absorption proceeds as follows. First, $\mathcal{T}_u$, $\mathcal{T}_r$ and $\mathcal{T}_g$ are initialised such that $\mathcal{T}_r = \emptyset$, $\mathcal{T}_u \cup \mathcal{T}_g = \mathcal{T}$ and $\mathcal{T}_u$ is unfoldable. This can be trivially achieved by setting $\mathcal{T}_g = \mathcal{T}$ and $\mathcal{T}_u = \mathcal{T}_r = \emptyset$, but it is usually best to try to maximise the number of definitional axioms in $\mathcal{T}_u$, and in particular to maximise the number of definitional axioms of the form $A \equiv D$ in $\mathcal{T}_u$ (Horrocks and Tobies, 2000b). Due to the uniqueness and acyclicity restrictions, however, there may be no unique maximal $\mathcal{T}_u$.

Next, the axioms in $\mathcal{T}_g$ are normalised by rewriting them as semantically equivalent axioms of the form $\top \sqsubseteq C$, i.e., $A \sqsubseteq B$ is rewritten as $\top \sqsubseteq C$, where $C = (B \sqcup \neg A)$. The concepts occurring in such axioms can be simplified using the techniques described in Section 3.1, and trivial axioms can be dealt with as follows:

- $\top \sqsubseteq \top$ is trivially satisfiable and can be removed from the TBox.
- $\top \sqsubseteq \bot$ is trivially unsatisfiable and leads directly to TBox unsatisfiability.

Finally, a range of rewriting rules can be applied to axioms in $\mathcal{T}_g$ in order to transform them into a suitable form, and then add them to either $\mathcal{T}_u$ or $\mathcal{T}_r$. These rules are repeatedly applied until either $\mathcal{T}_g$ is empty or no further rules are applicable. Note that it is important to first eliminate told cycles, as described in Section 3.2.3, otherwise application of the rewriting rules may not terminate.

Axiom transformation rules:

- $\top \sqsubseteq B \sqcup C$, where B is a named concept with $B \equiv D \in \mathcal{T}_u$, can be rewritten as $\top \sqsubseteq D \sqcup C$.
- $\top \sqsubseteq \neg B \sqcup C$, where B is a named concept with $B \equiv D \in \mathcal{T}_u$, can be rewritten as $\top \sqsubseteq \neg D \sqcup C$.
- $\top \sqsubseteq (D_1 \sqcap D_2) \sqcup C$ can be rewritten as two axioms $\top \sqsubseteq D_1 \sqcup C$ and $\top \sqsubseteq D_2 \sqcup C$.

Concept absorption:

- $\top \sqsubseteq \neg A \sqcup C$, where A is a named concept with $A \sqsubseteq D \in \mathcal{T}_u$, can be absorbed into $\mathcal{T}_u$ by removing $\top \sqsubseteq \neg A \sqcup C$ from $\mathcal{T}_g$ and adding $A \sqsubseteq C$ to $\mathcal{T}_u$.

Role absorption:

- $\top \sqsubseteq (\forall R.C) \sqcup D$ can be absorbed into $\mathcal{T}_r$ by removing it from $\mathcal{T}_g$ and adding $\top \sqsubseteq \forall\, \mathsf{Inv}(R).((\forall R.C) \sqcup D)$ to $\mathcal{T}_r$.
- $\top \sqsubseteq (\leq nR.C) \sqcup D$ can be absorbed into $\mathcal{T}_r$ by removing it from $\mathcal{T}_g$ and adding $\top \sqsubseteq \forall\, \mathsf{Inv}(R).((\leq nR.C) \sqcup D)$ to $\mathcal{T}_r$.

Nominal absorption:

- $\top \sqsubseteq C$, where $C = \neg(o_1 \sqcup \ldots \sqcup o_n) \sqcup D$, can be absorbed into $\mathcal{T}_u$ by removing $\top \sqsubseteq C$ from $\mathcal{T}_g$ and adding $o_i \sqsubseteq C$ to $\mathcal{T}_u$ for all $1 \leq i \leq n$.

– $\top \sqsubseteq C$, where $C = \neg(\exists R.(o_1 \sqcup \ldots \sqcup o_n)) \sqcup D$, can be absorbed into $\mathcal{T}_u$ by removing $\top \sqsubseteq C$ from $\mathcal{T}_g$ and adding $o_i \sqsubseteq \forall \mathsf{Inv}(R).C$ to $\mathcal{T}_u$ for some $1 \leq i \leq n$.

Note that there may be many different ways to apply these rewriting rules, some of which may eventually lead to different absorptions. Defining what constitutes a "good" absorption is still an open problem, but an absorption that leaves $\mathcal{T}_g$ empty is invariably better in practice than one which does not (Horrocks and Tobies, 2000a; Tsarkov and Horrocks, 2004).

4. Optimisations in Core Satisfiability Testing

The core of FaCT++ is its KB satisfiability algorithm, which implements a highly optimised version of the tableaux algorithm described in Section 2.2. But before introducing used optimisations, we remind in short "standard" implementation on the tableaux algorithm for testing concept satisfiability.

4.1. Standard Satisfiability Algorithm

It is easy to prove termination and correctness of the tableaux algorithm presented in Section 2.2, but practical applicability is limited. No practical ways to deal with non-deterministic rules are given. All this lead us to define "standard" satisfiability testing algorithm which is based on theoretical description from Section 2.2.

One point worth to note is that concept expressions does *not* usually transformed to NNF. Instead, as we described in Section 3.1, algorithms used simplified normal form together with an additional $\neg$-rule, which allows algorithms to find clashes faster.

Real-life algorithms usually keeps track of changes in the completion graph (i.e., which concepts were added to node labels, or creation new edges in a completion graph) and maintain some ordering of concepts that have to be expanded. As a the same time several expansion rules might be applicable, so some order of applying rules may be chosen.

The other thing that differs real algorithm from theoretical description is the way it deals with branching rules.

For every expansion of branching rule algorithm creates a *branching point*, in which it saves state of a reasoning process (including completion graph, chosen alternative in branching concept, etc), apply chosen decision and continue reasoning process. If later on clash occurs, algorithm *backtracks* to the latest branching point, restores state and apply next alternative. If all possible choices from given branching rule leads to clash, algorithm returns to the

previous branching point. If backtrack happens in the first created branching points, then building of a completion graph fails, and concept is unsatisfiable.

So, there exists two main sources of non-determinism in tableaux algorithm: choice of an expansion rule to apply, and choose an alternative in a branching rule. Most optimisations that are used in real implementations are targeted these two.

4.2. Ordering of Expansion Rules

Most systems use (modified) *top-down* approach for ordering expansion rules. In this approach generating rules (like $\geq$-rule and $\exists$-rule) are applied after all other rules. This comes from times where description logics were weaker and hence more tractable. For some subsets of $\mathcal{SHOIQ}$, like $\mathcal{SHF}$ (logic with simplified number restrictions and without inverse roles and nominals), such strategy might be used for creating algorithms with polynomial size (so-called *trace technique*). In short, algorithm builds the only branch of a completion tree (for such logics completion graph is tree-shaped), fully expand it and then discard fully expanded sub-tree in order to reuse space for the other branches.

Note that trace technique can not be used in presence of inverse roles, in particular because concepts that involve inverse roles may propagate information back from fully expanded subtrees. However, most of algorithms still use the same (accordingly modified) schema.

The FaCT++ system was designed with the intention of implementing DLs that include inverse roles, and of investigating new optimisation techniques, including new ordering heuristics.

Instead of the top-down approach, FaCT++ uses a *ToDo list* to control the application of the expansion rules. The basic idea behind this approach is that rules may become applicable whenever a concept is added to a node label. When this happens, a note of the node/concept pair is added to the ToDo list. The ToDo list sorts all entries according to some order, and gives access to the "first" element in the list. A given tableaux algorithm takes an entry from the ToDo list and processes it according to the expansion rule(s) relevant to the entry (if any). During the expansion process, new concepts may be added to node labels, and hence entries may be added to the ToDo list. The process continues until either a clash occurs or the ToDo list become empty.

In FaCT++ the ToDo list architecture is implemented as a priority queue. It is possible to set a priority for each rule type, and whenever entry would be added to the queue, it inserts after already existing entries with same of higher priorities. This means that if the $\exists$-rule (the generating rule that expands existential restrictions) has a low priority (say 0), and all other rules have

the same priority (say 1), then the expansion will be (modulo inverse roles) top-down and breadth first.

This architecture has a number of advantages when compared to the top-down approach. Firstly, it is applicable to a much wider range of logics, including the expressive logics implemented in modern systems, because it makes no assumptions about the structure of the graph (in particular, whether tree-shaped or not), or the order in which the graph will be constructed. Secondly, it allows for the use of more powerful heuristics that try to improve typical case performance by varying the global order in which different syntactic structures are decomposed; in a top-down construction, such heuristics can only operate on a local region of the graph—typically a single node.

Empirical analysis shows (Tsarkov and Horrocks, 2005b) that the best ordering for the expansion rules application is whether $\sqcup$-rule has the lowest priority, generating rules has second-lowest priority, and then all other rules has the same priority (except for o-rule and o?-rule, that should have higher priority in order to ensure termination of the algorithm).

4.3. DEPENDENCY-DIRECTED BACKTRACKING (BACKJUMPING)

Consider, for example, the following concept:

$$(C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \sqcap \exists R.(A \sqcap B) \sqcap \forall R.\neg A.$$

In a classic top-down architecture the disjunctions would be expanded before the existential. So the tableaux algorithm would first expand n disjunctions and then find a clash due to the $\exists-$rule.[4] In the case of normal (unoptimised) backtracking, 2^n choices of different disjunctions expansions would be done before the concept would be determined as unsatisfiable.

To avoid an exponential search during checking the satisfiability of C and similar concepts, a more sophisticated solution is required, and can be found by adapting a form of dependency directed backtracking called *backjumping*, which has also been used, e.g., in solving constraint satisfiability problems (Baker, 1995) and (in a slightly different form) in the HARP theorem prover (Oppacher and Suen, 1988).

Intuitively, backjumping works by labelling each concept C in the label of a node x with a dependency set $Dep_C(x)$ indicating the *branching points* (i.e., applications of the a branching rule) on which it depends. In case the completion tree contains some node x with $\{C, \neg C\} \in \mathcal{L}(x)$, we use $Dep_C(x)$ and $Dep_{\neg C}(x)$ to identify the most recent branching point b on which either C or $\neg C$ depends. The algorithm can then *jump* back to b over

[4] If the ToDo list algorithm is used with the priority of $\exists$ higher than priority of $\sqcup$, clash would be found after two expansion of $\exists$-rule. However, this example can be easily modified to have an exponential behaviour even in such a case.

intervening branching points *without* exploring any alternative branches (non-deterministic choices), and make a different non-deterministic choice which might not lead to the same closure condition being encountered. In case no such b exists, the closure did not depend on any non-deterministic choice, and the algorithm stops.

4.4. Optimisations of Disjunction Processing

4.4.1. *Boolean constraint propagation (BCP)*

As well as the standard tableau expansion rules described in Section 2, additional inference rules can be applied to the concept occurring in a node label, usually with the objective of simplifying them and reducing the number of $\sqcup$-rule applications. The most commonly used simplification, often called *Boolean Constraint Propagation* (BCP) (Freeman, 1995), is derived from SAT solvers, where it is usually used in conjunction with the Davis-Putnam procedure. The basic idea is to identify a disjunction $C_1 \sqcup \ldots \sqcup C_n \in \mathcal{L}(x)$ such that the negations of all but one of the C_j are already elements of $\mathcal{L}(x)$; when this is the case, the disjunction can be deterministically expanded by adding the relevant C_j to $\mathcal{L}(x)$. This amounts to applying the following inference rule

$$\frac{\neg C_1, \ldots, \neg C_n, C_1 \sqcup \ldots \sqcup C_n \sqcup C}{C}$$

to the concept in a node label.

Note that, as with the more sophisticated search techniques described above, careful consideration needs to be given to the dependencies of concepts added by such inference rules if they are to be used together with backjumping.

4.4.2. *Semantic Branching*

Assume expansion of the following concept (for simplicity, let disjunctions will be expanded before an existential; assume also, that C would be the first choice in all the disjunctions):

$$(C \sqcup D_1) \sqcap \ldots \sqcap (C \sqcup D_n) \sqcap \exists R.\forall R^{-}.\neg C.$$

On the 1st branching point C was added to a label of a node. All other disjunctions became expanded (expansion rule wouldn't run because part of a disjunction (namely, C) contains in a label of a node). Further expansion lead to a clash, because $\neg C$ would be propagated to a node after application of $\exists$-rule and $\forall$-rule. After backtracking, D_1 would be chosen from the 1st disjunction. But then C would be chosen from the 2nd disjunction, and the story continues. In that case, $3n$ expansion rules would be applied before clash-free completion graph would be obtained.

Semantic Branching (Giunchiglia and Sebastiani, 1996) allows algorithm to avoid such situation. During expansion of $C \sqcup D$ concept in the node x, it first tries to add C to $\mathcal{L}(x)$. If this attempt fails, next try would be made with adding $\{D, \neg C\}$ to $\mathcal{L}(x)$. This prevents following attempts to add the (failing) concept C to the label of x.

In an example given in the beginning of a section, after failed expansion of C, algorithm adds $\{\neg C, D_1\}$ to the node label. All other disjunctions will be expanded deterministically using BCP. In order to build completion graph, $n + 4$ expansion rules would be applied.

4.4.3. *Heuristics for Choosing Expansion Ordering*

It is well known that different order of expanding non-deterministic rule can result in huge (up to several orders of magnitude) difference in reasoning performance (Tsarkov and Horrocks, 2005b). Heuristics can be used to choose a "good" order in which to try the different possible expansions of such rules. In practise, this usually means using heuristics to select the way in which the $\sqcup$-rule is applied to the disjunctions in a node label; a heuristic function is used to compute the relative "goodness" of candidate concept.

When using the Davis-Putnam technique, the well known MOMS heuristic (Freeman, 1995) is often used to select the concept on which to branch; it tries to select concept that will maximise the effect of BCP and so minimise the number of non-deterministic choices needed in order to complete the completion graph (Horrocks, 2003). There is little evidence, however, that (a suitably adapted form of) this heuristic is effective with concepts, and even some evidence to suggest that interference with the backjumping optimisation makes it counter productive (Horrocks, 2003).

An alternative heuristic, whose design was prompted by this observation, tries to maximise the effect of backjumping by preferentially selecting concepts with low valued dependencies (Horrocks, 2003; Hladik, 2002). This heuristic has the added advantage that it can also be used to select the order in which successor nodes are expanded.

Usually several possible expansion-ordering heuristics can be used to choose the order in which to explore the different expansion choices offered by the non-deterministic $\sqcup$-rule. This ordering can be on the basis of the size, maximum quantifier depth, or frequency of usage of each of the concepts in the disjunction, and the order can be either ascending (smallest size, minimum depth and lowest frequency first) or descending. In order to avoid the cost of repeatedly computing such values, all the relevant statistics for each concept can be gathered as the knowledge base is loaded.

In (Sirin et al., 2005a) learning-based disjunct selection approach was proposed. At the very beginning all disjuncts has the same priority. Later on, after clash occurs because of disjunction expansion, the disjunct which was a cause

of a clash is penaltised. During consequent expansions, penaltised disjuncts got lower priority, according their penalty. This approach should work better for disjunctions on nominal nodes, because they are expanded large amount of time in similar manner, and wrong statically chosen ordering on them leads to significant slowdown.

4.4.4. *Combined Expansion Rule for Disjunction*
The above optimisations for disjunction processing can be described as the following complex rule for disjunctions:

- For the initial expansion of a disjunction, determine the type of the disjunction, given by the concept expression $C_1 \sqcup \ldots \sqcup C_n$.:
  1. If $C_i \in \mathcal{L}(x)$ for some i, then there is no need to expand the disjunction.
  2. Otherwise, determine the set $C' = \{C_{i_1}, \ldots, C_{i_k}\} \subset \{C_1, \ldots, C_n\}$ such that $\neg C_{i_j} \notin \mathcal{L}(x)$ for all j. All elements of set C' could be added to a label.
  3. Choose the (heuristical) weight function $Heu(C)$ such that if $Heu(C) < Heu(C')$ then C should be chosen to expand earlier than C'.
  4. Initialize the set of processed disjuncts $C_p = \emptyset$.
- For the general expansion step:
  1. Choose element c from the ordered set C' such that $\forall c' \in C' : Heu(c) \leq Heu(c')$.
  2. If semantic branching is in use, add $\neg c'$ to $\mathcal{L}(x)$ for all $c' \in C_p$.
  3. Set $C_p = C_p \cup \{C\}$ and $C' = C' \backslash \{C\}$.
  4. If $C' \neq \emptyset$, then create a branching point.
  5. Add C to $\mathcal{L}(x)$.

4.4.5. *Save/restore optimisation*
In order to perform backtracking a tableaux algorithm must be able to save its internal state before branching and restore it in case of failure detection. Besides other things, state of the completion graph (including node labels, edges labels, etc) should be saved.

A naive approach would be to save everything after branching and restore everything after backjumping. But in realistic applications, the completion graph may contain hundreds or thousands of nodes, and only some of them are changed between branching decisions.

Moreover, when using the ToDo List approach, branching operations are grouped together, and so they are often expanded in all a row. In this case, operations usually change just a small part of the completion graph while the rest of it is unchanged between different branching points. All this makes naive approach inefficient in realistic applications.

4.4.5.1. *Lazy saving* The goal of the *lazy saving* optimisation is to minimise saving of the nodes of the completion graph. Instead of saving the whole completion graph on every branching point, lazy saving implements a "save-on-demand" approach.

In this approach every node of the completion graph contains an additional field—the branching level of the last change. Whenever a node would be changed (this includes additions to node label, adding new incoming/outgoing edges, merging nodes, changing blocked status, etc.) the algorithm checks the last saved level of the node. If the current branching was made after the last save of the node, then the node's information is outdated. The algorithm saves the state of the node, which now can be safely modified.

In addition to obvious gains, restoring of completion graph is also slightly simplified. In case of restoring from level n back to level m, if the node wasn't changed since m, there is nothing to do with respect to restore.

In large completion graphs usually only a small number of nodes is changed at the same branching level. This is especially true in case where priority of OR operation is the lowest. So, avoiding unnecessary saving of huge amounts of nodes leads to increasing performance. However, the check of the necessity of saving must be done on every updating. For lazy saving to be cost-effective this check must should be cheap. In the KBs with a small number of branching concepts this check may consume more time than would be gained as a result of an optimisation.

4.4.5.2. *Lazy restoring* The similar idea, restore-on-demand, may come to one's mind. Instead of doing restoration once per backjumping for all nodes of the completion graph, it is possible to check before accessing nodes whether it is necessary to restore the state of the node. If the node appears to be out of date, it is necessary to restore it before taking any information from the node.

However, this approach is different from the lazy saving in the following:

- Lazy restoring is a check-on-read instead of a check-on-write technique. As there are many more reads than writes to a completion graph ($\forall-$rule application, blocking checks etc), many more resources would be used for these checks.
- In lazy saving, after backjumping new branching points can reuse old labels of branching levels. This is because there is no one entry (after

backjumping) which uses labels. This is not true in case of lazy restoring, as there are nodes that still use the old branching labels. In some cases, it might be necessary to keep and (probably) maintain the whole tree of branching decisions.

– The number of restores is usually just a small fraction of the number of saves. In our experiments on realistic KBs restores were between 0.5 to 5% as frequent as saves. This indicates that lazy saving would provide at least 20 times more benefits than lazy restoring.

4.5. COMBINED GENERATING RULES

After creating a new node or changing role labels all universal restriction rules must be re-applied. However, it is usually better to re-apply them immediately during generating rule expansion.

For example, assume concept $\exists R.C$ was expanded in node x and lead to the creation of node y with $C \in \mathcal{L}(y)$. As usually a $\exists$-rule is expanded later than a $\forall$-rule, all $\forall$-concepts appearing in $\mathcal{L}(x)$ were already expanded. But now the new R-neighbour of x immediately appears, so all $\forall$-concepts in $\mathcal{L}(x)$ must be expanded again. Some work can be saved if the algorithm can ensure that *all* such concepts were initially expanded before *any* $\exists$-concept. In this case, it is enough to re-expand only those $\forall S.D$ concepts, for which $R \sqsubseteq^* S$.

4.6. CACHING SAT STATUS

4.6.0.3. *Summary:* Instead of creating new node using (combined) generating rule, it is possible to check SAT status of new node by trying to merge (cached) models of concepts in new node. Is such models can be merged, then there is no need to expand corresponding subtree.

Assume no inverse roles and no nominals are in KB and top-down technique is used. I.e., The expansion rules are sorted with generating rules have the lowest priority.

In this case, all information from predecessors is added to a node label before it is processed. This means that, when a given node has been fully expanded (i.e., the expansion rules have been exhaustively applied to it), a successor node y with $\mathcal{L}(y) = \{C_1, \ldots, C_n\}$ can be treated as an independent problem, equivalent to testing the satisfiability of $C_1 \sqcap \ldots \sqcap C_n$.

A completion tree may contain many such nodes, and the labels of nodes tend to be quite similar. For some concepts, this may result in the same sub-problem being solved again and again. In order to avoid this, it is possible to cache and re-use the results of such sub-problems. The usual technique is to use a hash table to store the satisfiability status of node labels (i.e., sets of

concepts treated as a conjunction). Before applying any expansion rules to a new node x, the cache is interrogated to determine if the satisfiability status of $\mathcal{L}(x)$ is already known. If it is known, then the result can be used without further expansion, i.e., $\mathcal{L}(x)$ can be treated as though it were either $\{\bot\}$ (for unsatisfiable) or $\{\top\}$ (for satisfiable). If the satisfiability status of $\mathcal{L}(x)$ is not known, then $\mathcal{L}(x)$ is added to the cache, and its status set to satisfiable if a complete and open completion tree rooted in x can be constructed, and to unsatisfiable otherwise. The procedure of determining satisfiability status for given set of concepts is well-known and was described in a great details in several papers (Horrocks, 2003; Haarslev and Möller, 2001a).

Since the satisfiability of a set of concepts L implies the satisfiability of each subset of L, and the unsatisfiability of a set of concepts L implies the unsatisfiability of each superset of L, this basic idea can be extended to check for satisfiable supersets of $\mathcal{L}(x)$ and unsatisfiable subsets of $\mathcal{L}(x)$. However, this requires a considerably more sophisticated data structure if cache operations are to be efficient (Hoffmann and Koehler, 1999; Giunchiglia and Tacchella, 2000).

Apart from the problem of the storage required for the cache, another more subtle disadvantage of caching is that, in the case where the cache returns "unsatisfiable" for $\mathcal{L}(x)$, there is no information about the cause of the unsatisfiability that can be used to derive the dependency information required for backjumping. Backjumping can still be performed by combining the dependency sets of all of the concepts in $\mathcal{L}(x)$, but this is likely to overestimate the set of branching points on which the unsatisfiability depends.

In presence of inverse roles or nominals, however, this technique can not be directly used. If SAT checking involves nominals, they can be referred from different nodes of completion graph, so new concepts may be propagated to already cached node. In case of inverse roles, information may be propagated from the node label to it parent, so just checking SAT status of the child is not enough. Latest publications (Yu Ding, 2005) propose some directions to use of cache in presence of inverse roles but logic used there is much weaker than $\mathcal{SHOIQ}$.

5. Optimisations in Classification

Classification is the process of establishing partial order $\leq$ on the set of named concepts in KB w.r.t. subsumption relation, i.e. $C \leq D \iff C \sqsubseteq D$. This order is called *hierarchy*, or *taxonomy* of concepts. Classification is one of the main operations, performing by a reasoner for simplify following subsumption queries.

Traditionally, this partial order relation is built iteratively. It is initialised with trivial relation $\bot \leq \top$ and on every iteration one new concept name C

is added to a hierarchy. For every concept name C to be added to hierarchy, the set of *parents* (i.e., already classified immediate subsumers) and the set of *children* (i.e., already classified immediate subsumees) is being defined. Sets of parents and children uniquely identify place of C in the taxonomy.

Set of parents (children) is defined by a procedure called *top-down* (resp. *bottom-up*) search. These two procedures are very similar, so we describe only one of them in details.

Top-down search phase for concept C involves breadth-first search of subsumers of C starting from $\top$. Once it is determined that $C \sqsubseteq D$ the algorithms search subsumers between children of D. Concepts that are subsumers of C but none of its children is subsumer of C became parents of C.

This algorithm was first described in (Baader et al., 1994). This paper also adds some optimisation that allows significantly reduce search space.

5.1. TAXONOMY CREATION ORDER

Concept name C *directly uses* concept name D if D occurs in the definition of C. The relation *uses* is a transitive closure of directly uses. If C uses D then D comes before C in so-called *definition order*. Assume that KB is classified in definition order, i.e. a concept name is not classified until all the named concepts it uses are classified. In (Baader et al., 1994) it was shown, that in this case bottom-up search phase can be omitted for primitive concepts (the only child for such concept would be $\bot$ node). Easy to see that it is true for $\mathcal{SHOIQ}$ TBox $\mathcal{T} = \mathcal{T}_u \cup \mathcal{T}_r \cup \mathcal{T}_g$ with $\mathcal{T}_g = \emptyset$, i.e. without GCIs. If $\mathcal{T}_g \neq \emptyset$, however, it is not true. Assume TBox $\mathcal{T} = \mathcal{T}_u \cup \mathcal{T}_g$ with

$$\mathcal{T}_u = \{C_1 \sqsubseteq D, C_2 \sqsubseteq D\}, \mathcal{T}_g = \{\top \sqsubseteq C_1 \sqcap C_2\}.$$

In this TBox concept D should be classified equal to $\top$, i.e. have $\top$ both as a parent and children. If bottom-up phase would be omitted, however, algorithm would be unable to find $\top \sqsubseteq D$ subsumption and hence gives wrong results.

In (Haarslev and Möller, 2001a) this approach was modified in order to allow to skip bottom-up phase in more cases. Authors propose to use *directly refers to* relation, which is similar to directly uses, but with references occurring in scope of quantifiers are not considered. Again *refers to* is transitive closure of directly refers to, and this relation induces a partial order relation on sets of concept names. Total order produced by this relation is called *quasi-definition order*.

Usage of quasi-definition order instead of definition order suppose that subsumption relation between primitive concepts can't be derived via quantifiers. This is not true in presence of inverse roles. For the KB

$$C \sqsubseteq D, D \sqsubseteq \exists R.\forall R^{-}.C$$

quasi-definition order will set $D \leq C$, while semantically $C \equiv D$, i.e. both C and D should be classified into the same node of taxonomy.

5.2. TOLD SUBSUMERS AND DISJOINTS

Various optimisations are used in order to minimise the number of subsumption tests needed in each phase. For example, when adding a concept C to the hierarchy, a top-down breadth first traversal is used that only checks if D subsumes C when it has already been determined that C is subsumed by all the concepts in the hierarchy that subsume D. The structure of TBox axioms is also used to compute a set of *told subsumers* of C (i.e., trivially obvious subsumers). See Section 3.2.3 for the formal definition of a told subsumer. As subsumption is immediate for told subsumers, no tests need to be performed w.r.t. these concepts. In order to maximise the benefit of this optimisation, all of the told subsumers of a concept C are classified *before* C itself is classified.

The told subsumer optimisation can be used to approximate the position of C in the hierarchy: all of its told subsumers can be marked as subsumers of C. The most specific concepts in this set of marked concepts are then candidates to be parents of C. In the standard algorithm, however, it is necessary to check (recursively) if the children of these concepts are also subsumers of C. This can be costly in the case where one of the told subsumers has a very large number of children. When it has been determined for some subsumer D of C that none of the children of D subsume C, then we know that D is a parent of C.

At the end of the top-down phase we will have computed the set of parents of C; all of the concepts in this set, along with all their super-concepts, are subsumers of C; all other concepts are non-subsumers of C. The next step is to determine the set of children of C (as mentioned above, this step can be omitted for a primitive concept when concepts have been classified in definitional order (Baader et al., 1994)). This phase is very similar to (the reverse of) the top-down one, and we won't describe it here—interested readers can refer to (Baader et al., 2003) for full details.

In addition to use told subsumers to propagate positive subsumption information, it is possible to use *told disjoints* to propagate negative subsumption information (Haarslev and Möller, 2001a). Concept D is *told disjoint* with concept C if definition of C looks like $C \sqsubseteq \neg D \sqcap \ldots$. Having told disjoint information, it is possible to immediately state non-subsumption (and propagate this information if necessary).

5.3. REDUCING NUMBER OF SUBSUMPTION TESTS

In some cases, it is possible to replace expensive subsumption test with another (incomplete, but cheap) tests.

As one of such methods pseudo-model merging technique described in details in Section 4.6 can be used. If models for D and $\neg C$ can be merged, then concept $\neg C \sqcap D$ is satisfiable and then subsumption $C \sqsubseteq D$ does not holds. Unfortunately, as noted in Section 4.6, this test can't be applied directly in presence of nominals in the TBox.

Other methods were developed to reduce number of checks while traversing taxonomy during top-down or bottom-up phase. As these two phase are very similar, we will talk about optimisations w.r.t. the top-down one.

Some labelling optimisations were proposed in (Baader et al., 1994) in addition to the told subsumer one. When D is going to be checked as a subsumer of classifying concept C, it is first checked whether all subsumers of D are subsumers of C. The opposite one is non-subsumption propagation: if $C \sqsubseteq D$ subsumption does not holds, so do $C \sqsubseteq D'$ for all D' that are subsumees of D. These two optimisations does not work well together; in (Baader et al., 1994) it was shown that the former one is more efficient in general than the latter.

In case of wide (and shallow) taxonomies, one taxonomy node (with label D) may have tens and hundreds children D_j. If such a node is labelled as a subsumer of classifying concept C, then large amount of subsumption tests $C \sqsubseteq D_j$ should be made. In this case, in (Haarslev and Möller, 2001a) so-called *clustering technique* is proposed. For concepts $D_1 \dots D_\theta$ new virtual concept $D' \equiv D_1 \sqcup \dots \sqcup D_\theta$ is inserted into taxonomy. And instead of checking $C \sqsubseteq D_j$ first subsumption $D' \sqsubseteq C$ is checked. This check is done using model merging technique, because model for $\neg D'$ is just a union of negated names for primitive concepts $D_1, \dots, D_\theta$. In addition, it is possible to have several virtual concepts associated with the same parent node.

Another way to reduce number of subsumption tests is to use notion of completely defined concepts, that we are going to describe in more details below.

5.4. Completely Defined Concepts

Given a TBox $\mathcal{T}$, a primitive concept C is said to be *completely defined* w.r.t. $\mathcal{T}$ when, for the definition $C \sqsubseteq C_1 \sqcap \dots \sqcap C_n$ in $\mathcal{T}$, it holds that:

1. For all $1 \leqslant i \leqslant n$, C_i is a primitive concept.
2. (minimality) There exist no i, j such that $1 \leq i, j \leq n$ and C_j is a told subsumer of C_i.

When the TBox is obvious from the context we will talk about completely defined concepts without reference to a TBox.

If we assume a cycle-free TBox containing only CD concepts and no GCIs, then the classification process is very simple. In fact, we don't need

to perform any subsumption tests at all: the position of every concept in the hierarchy is *completely defined* by its told subsumers. If concepts are processed in definitional order, then when a concept C is classified, where C is defined by the axiom $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$, the parents of C are $C_1, \ldots, C_n$, and the only child of C is $\bot$. Note that every concept in such a taxonomy is satisfiable, because there is no use of negation.

Correctness of the above method for classification of a cycle-free TBox which contains only completely-defined concepts and no general axioms was proved in (Tsarkov and Horrocks, 2005a).

This fact is, however, of very little practical value due to the very stringent conditions on the structure of the TBox. In the following, we will show how the basic technique can be made more useful by weakening some of these conditions.

5.4.1. *Primitivity*
In general, a CD concept should not have non-primitive concepts in its definition. This is because when the hierarchy already includes non-primitive concepts (which will be the case given definitional order classification) the bottom-up phase can not be omitted, and the CD method could therefore lead to incorrect results. Assume, e.g., a TBox

$$\{C \sqsubseteq C_1 \sqcap C_2 \sqcap C_3, \quad C' \equiv C_1 \sqcap C_2\}. \tag{1}$$

Using the CD classification approach, C will be classified under C_1, C_2 and C_3, whereas it should be classified under C' and C_3.

5.4.2. *Minimality*
Non-minimal concepts may occur as a result of badly designed KBs, improper usage of Domain constraints and/or due to absorption of GCIs. The minimality check may, however, be removed from the definition of CD concepts provided that we check for any non-minimal concepts at classification time, i.e., we check if each C_i in a definition $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$ is really a parent of C (i.e., has no children that are subsumers of C). This check is relatively cheap, and is already implemented as part of the standard classification algorithm (see Section 5.4.5), where it is used to check which of the told subsumers of a concept C are possible parents of C.

5.4.3. *Non-CD Concepts*
CD-classification would be of limited interest if its applicability were limited to KBs consisting entirely of CD concepts. This is because most "interesting" KBs, including most KBs designed using DL based ontology languages, will contain concept constructors other than conjunction, and this will lead to some concepts being non-CD; in its current form the CD approach would, therefore, be useless.

On the other hand, almost all TBoxes will contain *some* CD concepts. In this case, it may be possible to split the TBox into two parts—a CD part (i.e., containing only CD concepts) and a non-CD part—and use the CD algorithm only for the CD part.

Note that such a split will not introduce any problems if the CD part of the classification is performed first—in fact the classification of the CD part is independent of the non-CD part of the TBox, because the definitions of CD concepts only refer to other CD concepts.

In the TBox 1 above, for example, concepts C_1, C_2, C_3 and C will be in the CD part, and C' in the non-CD part. After CD-classification C will have 3 parents, and standard algorithm then insert C' with parents C_1, C_2 and child C.

5.4.4. *General Axioms*

It is easy to see that, in the general case, the CD approach cannot be used in the presence of GCIs. Consider, for example, a TBox $\mathcal{T} = \mathcal{T}_u \cup \mathcal{T}_g$ with

$$\mathcal{T}_u = \{C \sqsubseteq \top, A \sqsubseteq D, B \sqsubseteq D\}, \mathcal{T}_g = \{\top \sqsubseteq A \sqcup B\}.$$

In this case, the CD algorithm classifies C under $\top$, whereas it should be classified under D. The CD approach is, therefore, applicable only if all GCIs in the TBox are absorbed (i.e., $T_g = \emptyset$) using the techniques described in Section 3.2.5.

5.4.5. *Two-stage Approach Using CD*

A two-stage CD classification algorithm has been implemented in our FaCT++ reasoner. After preprocessing, as described in Section 3.1, the TBox is classified using the following procedure:

1. If the TBox does not contain any GCIs, mark some concepts as CD. Namely, $\top$ is marked as CD; a primitive concept C is marked as CD if it has the definition $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$ and every C_i is marked CD; a non-primitive concept D is marked CD if it has definition $D = C$ and C is marked CD.

2. If the TBox contains concepts marked as CD, then run the CD-classifier. The CD classifier works only on those concepts that are marked CD, processing them in definitional order. For each such concept C, the steps it performs are as follows:

   a) If C is a synonym of some already classified concept D, then insert C at the same place as D.

   b) If C has definition $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$, concepts $C_1, \ldots, C_n$ are candidates to be parents of C.

c) For every candidate C_i, check whether it is redundant, i.e. whether C_i has a child that is an ancestor of a C. This can be done by labelling all ancestors of candidate concepts: labelled candidates will be redundant. Remove redundant candidates from the list of candidates.

d) Insert C into the taxonomy with the remaining candidates as parents and $\bot$ as the only child.

3. Classify the remaining concepts using the standard classification algorithm.

6. Discussion

6.1. NOTES ON EVALUATION

We does not plan to put any evaluation on this paper. Most of optimisations mentioned here are described in other papers, which usually contain some empirical evaluation. In this section we just summarise all described optimisations and mark them with respect to their influence on overall performance of a DL reasoner.

6.1.1. *Preprocessing Optimisations*

Note that all preprocessing optimisation takes at most polynomial time to run, while they hardly hurts performance of further reasoning. This makes them good in almost any cases.

Lexical Normalisation. It helps a lot in founding clashes earlier, so lazy unfolding optimisation performs much better.

Lexical Simplifications. Most of them are so simple, that takes literally no time to apply. Some of simplifications, defined in Section 3.1 though are more costly and gives effect only in very special cases. It makes sense then to apply such transformation to concept expressions that are *known* to be complex and are likely to follow under this cases (like GCIs and absorbed concept definitions).

Lazy Unfolding. Easy to see that use of $\sqsubseteq$-rule in realistic DL reasoners is a nonsense due to huge amount of disjunctions it introduces. Thus there is no reasons not to use lazy unfolding.

Synonym Replacement. This optimisation has linear complexity and works well together with told cycle elimination and some special kind of KBs (e.g. those which were obtained by merging several KBs together).

Told Cycle Elimination. While not very useful by itself and rarely applicable in a real-life KBs, this optimisation costs nothing (as told subsumers are necessary for classification) and if applicable, allows more optimisation (like absorption and definition-order classification) to run.

Redundant Subsumption Elimination. Again, it is hard to see any use of such an optimisation for a nornal KB, but it is useful to run it after told cycle elimination (and in general, in presence of cycles of length 1).

Absorption. As shown in all optimisation paper on this topic (see (Horrocks and Tobies, 2000b; Tsarkov and Horrocks, 2004)), absorption is crucial optimisation for DL reasoners. Is is though hard to define the best absorption (and it is even hard to say what does it mean), but *any* absorption, obtained from a TBox is better in terms of classification time.

6.1.2. *Core Satisfiability Optimisations*

For expressive DLs like $\mathcal{SHOIQ}$ reasoning is intractable. This means that reasoning procedures have at least exponential complexity. The task of optimisations, thus, is to reduce complexity of a reasoning of a frequent tasks (classes of tasks).

TODO List Architecture. Using such an optimisation makes reasoner more flexible. In (Tsarkov and Horrocks, 2005a) we shows that in most cases giving branching rules the lowest priority is better. Note that the standard top-down approach gives the lowest priority to generating rules. Moreover, decision procedure for $\mathcal{SHOIQ}$ requires some ordering on expansion rule application in order to ensure termination (Horrocks and Sattler, 2005). This requirement is easily incorporated into TODO list schema.

Note that TODO list-based reasoner (unlike the top-down approach) can not stay in polynomial space anymore. Though, reasoning with inverse roles required exponential space, so any reasoner for $\mathcal{SHOIQ}$ would lost this ability.

BCP. This simple optimisation might transform non-deterministic operation into a deterministic one. In any case, it does not harm reasoning process as it prevents reasoner to choose disjuncts that immediately leads to clash.

Semantic Branching. As mentioned in (Horrocks and Patel-Schneider, 1999), using this optimisation gives significant increase of performance on artificial tests. Our experiments with real applications shows that using this optimisations gives very small, but almost always positive effect.

Heuristics for Disjunction. As shown in (Tsarkov and Horrocks, 2005a), choosing improper heuristic might lead to completely impractical reasoning (the difference in some cases is more then three orders in magnitude). Unfortunately, we don't have stable way of choosing proper heuristic, so it is a part of future research.

Note also, that some heuristics (like MOMS), interacts badly with backjumping (which is one of the most beneficial optimisation techniques in SAT testing).

Backjumping. Another optimisation that is crucial for DL reasoning. Even the necessarity to build and carefully maintains dependency sets is overpowered by benefits that are obtained from this optimisation.

Lazy Saving. This optimisation works well in KBs that have complex structure (and, hence large completion graphs appears during SAT tests). This is also very important if TBox contains large number of nominals, as usually all (or large part) of them should always exists in a completion graph (Sirin et al., 2005a).

Caching SAT Status. This optimisation is applicable only in absence of inverse roles. But in this case it shows amazing improvement in reasoning, significantly reducing time of building completion graph.

6.1.3. *Classification Optimisations*

Note that, while all classification optimisations saves (at most) quadratic amount of necessary subsumption tests, each test might take exponentially many operations by itself, which lead to significant improvements.

Taxonomy Creation Order.

Told Subsumer and Disjoints.

Caching Subsumptions. All these optimisations works well when applicable, reducing number of performed subsumption tests in the price of (at most) polynomial actions. They should be used every time, as experiments (Baader et al., 1994; Haarslev and Möller, 2001a) shows benefits of them in both artificial and natural KBs.

Completely Defined Classification. This optimisation, as it shown in (Tsarkov and Horrocks, 2005b), gives the best results on large KBs with very simple structure. At the same time, it does not require any extra actions, and does not hurts general KBs, as most of them contains (large or small) part that can be dealt with more optimal than using standard algorithm.

References

Baader, F., D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider (eds.): 2003, *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Baader, F., E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich: 1994, 'An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or: Making KRIS get a move on'. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management* **4**, 109–132.

Baader, F. and U. Sattler: 2001, 'An Overview of Tableau Algorithms for Description Logics'. *Studia Logica* **69**(1), 5–40.

Baker, A. B.: 1995, 'Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results'. Ph.D. thesis, University of Oregon.

Calvanese, D., G. De Giacomo, and M. Lenzerini: 1998a, 'On the Decidability of Query Containment under Constraints'. In: *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*. pp. 149–158.

Calvanese, D., G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati: 1998b, 'Description Logic Framework for Information Integration'. In: *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*. pp. 2–13.

Fensel, D., F. van Harmelen, I. Horrocks, D. L. McGuinness, and P. F. Patel-Schneider: 2001, 'OIL: An Ontology Infrastructure for the Semantic Web'. *IEEE Intelligent Systems* **16**(2), 38–45.

Freeman, J. W.: 1995, 'Improvements to Propositional Satisfiability Search Algorithms'. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

Giunchiglia, E. and A. Tacchella: 2000, 'A Subset-Matching Size-Bounded Cache for Satisfiability in Modal Logics'. In: *Proc. of the 4th Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX 2000)*. pp. 237–251, Springer.

Giunchiglia, F. and R. Sebastiani: 1996, 'Building Decision Procedures for Modal Logics from Propositional Decision Procedures—the Case Study of Modal K'. In: *Lecture Notes in Artificial Intelligence*, Vol. 1104 of *Lecture Notes in Artificial Intelligence*. pp. 583–597, Springer.

Haarslev, V. and R. Möller: 2000, 'Expressive ABox Reasoning with Number Restrictions, Role Hierarchies, and Transitively Closed Roles'. In: *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*. pp. 273–284.

Haarslev, V. and R. Möller: 2001a, 'High Performance Reasoning with Very Large Knowledge Bases: A Practical Case Study'. In: *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*. pp. 161–168.

Haarslev, V. and R. Möller: 2001b, 'Optimizing Reasoning in Description Logics with Qualified Number Restrictions'. In: *Proc. of the 2001 Description Logic Workshop (DL 2001)*. pp. 142–151, CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-49/`.

Hladik, J.: 2002, 'Implementation and Optimisation of a Tableau Algorithm for the Guarded Fragment'. In: U. Egly and C. G. Fermüller (eds.): *Proceedings of the International Conference on Automated Reasoning with Tableaux and Related Methods (Tableaux 2002)*, Vol. 2381 of *Lecture Notes in Artificial Intelligence*. Springer.

Hoffmann, J. and J. Koehler: 1999, 'A New Method to Index and Query Sets'. In: *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*. pp. 462–467.

Horrocks, I.: 1997, 'Optimising Tableaux Decision Procedures for Description Logics'. Ph.D. thesis, University of Manchester.

Horrocks, I.: 1998, 'Using an Expressive Description Logic: FaCT or Fiction?'. In: *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*. pp. 636–647.

Horrocks, I.: 2003, 'Implementation and Optimisation Techniques'. In: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider (eds.): *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, pp. 306–346.

Horrocks, I. and P. F. Patel-Schneider: 1998, 'DL Systems Comparison'. In: *Proc. of the 1998 Description Logic Workshop (DL'98)*. pp. 55–57, CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-11/`.

Horrocks, I. and P. F. Patel-Schneider: 1999, 'Optimizing Description Logic Subsumption'. *J. of Logic and Computation* **9**(3), 267–293.

Horrocks, I. and P. F. Patel-Schneider: 2003, 'Reducing OWL Entailment to Description Logic Satisfiability'. In: D. Fensel, K. Sycara, and J. Mylopoulos (eds.): *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*. pp. 17–29, Springer.

Horrocks, I., P. F. Patel-Schneider, and F. van Harmelen: 2002, 'Reviewing the Design of DAML+OIL: An Ontology Language for the Semantic Web'. In: *Proc. of the 18th Nat. Conf. on Artificial Intelligence (AAAI 2002)*. pp. 792–797, AAAI Press.

Horrocks, I., P. F. Patel-Schneider, and F. van Harmelen: 2003, 'From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language'. *J. of Web Semantics* **1**(1), 7–26.

Horrocks, I. and U. Sattler: 2002, 'Optimised Reasoning for $\mathcal{SHIQ}$'. In: *Proc. of the 15th Eur. Conf. on Artificial Intelligence (ECAI 2002)*. pp. 277–281.

Horrocks, I. and U. Sattler: 2005, 'A Tableaux Decision Procedure for $\mathcal{SHOIQ}$'. In: *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*.

Horrocks, I., U. Sattler, and S. Tobies: 1999, 'Practical Reasoning for Expressive Description Logics'. In: H. Ganzinger, D. McAllester, and A. Voronkov (eds.): *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*. pp. 161–180, Springer.

Horrocks, I., U. Sattler, and S. Tobies: 2000, 'Reasoning with Individuals for the Description Logic $\mathcal{SHIQ}$'. In: D. McAllester (ed.): *Proc. of the 17th Int. Conf. on Automated Deduction (CADE 2000)*, Vol. 1831 of *Lecture Notes in Computer Science*. pp. 482–496, Springer.

Horrocks, I. and S. Tobies: 2000a, 'Optimisation of Terminological Reasoning'. In: *Proc. of the 2000 Description Logic Workshop (DL 2000)*. pp. 183–192.

Horrocks, I. and S. Tobies: 2000b, 'Reasoning with Axioms: Theory and Practice'. In: *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*. pp. 285–296.

Kalyanpur, A., B. Parsia, and J. Hendler: 2005, 'A Tool for Working with Web Ontologies'. *International Journal on Semantic Web and Information Systems* **1**(1), 36–49.

Knublauch, H., R. Fergerson, N. Noy, and M. Musen: 2004, 'The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications'. In: S. A. McIlraith, D. Plexousakis, and F. van Harmelen (eds.): *Proc. of the 2004 International Semantic Web Conference (ISWC 2004)*. pp. 229–243, Springer.

Lutz, C.: 1999, 'Complexity of Terminological Reasoning Revisited'. In: *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, Vol. 1705 of *Lecture Notes in Artificial Intelligence*. pp. 181–200, Springer.

Massacci, F.: 1999, 'TANCS Non Classical System Comparison'. In: *Proc. of the 3rd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'99)*, Vol. 1617 of *Lecture Notes in Artificial Intelligence*.

McGuinness, D. L. and J. R. Wright: 1998, 'An Industrial Strength Description Logic-based Configuration Platform'. *IEEE Intelligent Systems* pp. 69–77.

Oppacher, F. and E. Suen: 1988, 'HARP: A Tableau-Based Theorem Prover'. *J. of Automated Reasoning* **4**, 69–100.

Pan, Z.: 2005, 'Benchmarking DL Reasoners Using Realistic Ontologies'. In: *Proc. of the OWL: Experiences and Directions Workshop*.

Protégé: 2003, '`http://protege.stanford.edu/`'.

Rector, A.: 2003, 'Medical Informatics'. In: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider (eds.): *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, pp. 415–435.

Schmidt-Schauß, M. and G. Smolka: 1991, 'Attributive Concept Descriptions with Complements'. *Artificial Intelligence* **48**(1), 1–26.

Sirin, E., B. C. Grau, and B. Parsia: 2005a, 'Optimizing Description Logic Reasoning for Nominals: First Results'. Technical report, University of Maryland Institute for Advanced Computes Studies (UMIACS), 2005-64. Available online at http://www.mindswap.org/papers/OptimizeReport.pdf.

Sirin, E., B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz: 2005b, 'Pellet: A Practical OWL-DL Reasoner'. Submitted for publication to Journal of Web Semantics.

Stevens, R., C. Goble, I. Horrocks, and S. Bechhofer: 2002, 'Building a bioinformatics ontology using OIL'. *IEEE Transactions on Information Technology in Biomedicine* **6**(2), 135–141.

Tsarkov, D. and I. Horrocks: 2004, 'Efficient Reasoning with Range and Domain Constraints'. In: *Proc. of the 2004 Description Logic Workshop (DL 2004)*. pp. 41–50.

Tsarkov, D. and I. Horrocks: 2005a, 'Optimised Classification for Taxonomic Knowledge Bases'. In: *Proc. of the 2005 Description Logic Workshop (DL 2005)*.

Tsarkov, D. and I. Horrocks: 2005b, 'Ordering Heuristics for Description Logic Reasoning'. In: *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*.

Wolstencroft, K., A. Brass, I. Horrocks, P. Lord, U. Sattler, R. Stevens, and D. Turi: 2005, 'A Little Semantic Web Goes a Long Way in Biology'. In: *Proc. of the 2005 International Semantic Web Conference (ISWC 2005)*. Springer.

Yu Ding, V. H.: 2005, 'Towards Efficient Reasoning for Description Logics with Inverse Roles'. In: *Proceedings of the 2005 International Workshop on Description Logics (DL-2005), Edinburgh, Scotland, UK, July 26-28*. pp. 208–215.

Appendix I:

"System Description: FaCT++1.0" by Dmitry Tsarkov and Ian Horrocks. Enclosed with the full electronic version of this report as the file FaCT++.pdf.

System Description: FaCT++ 1.0

Dmitry Tsarkov and Ian Horrocks

Department of Computer Science
The University of Manchester
Manchester, UK
{tsarkov|horrocks}@cs.man.ac.uk

Abstract. This is a system description of the Description Logic reasoner FaCT++ 1.0. The reasoner is modelled on the well-known FaCT reasoner, but employs a different system architecture and has some new features and optimisations.

1 Introduction

Description Logics (DLs) are a family of logic based knowledge representation formalisms [1]. Although they have a range of applications, they are perhaps best known as the basis for widely used ontology languages such as OIL, DAML+OIL and OWL [10].

A key motivation for basing ontology languages on DLs is that DL systems can then be used to provide computational services for ontology tools and applications [12, 13]. The increasing use of ontologies, along with increases in their size and complexity, brings with it a need for efficient DL reasoners. Given the high worst case complexity of the satisfiability/subsumption problem for the DLs in question (at least ExpTime-complete), optimisations that exploit the structure of typical ontologies are crucial to the viability of such reasoners.

FaCT++ is a new DL reasoner designed as a platform for experimenting with new tableaux algorithms and optimisation techniques.[1] It incorporates most of the standard optimisation techniques, including those introduced in the FaCT system [7], but employs a new "ToDo list" architecture that is better suited to more complex tableaux algorithms (such as those used to reason with OWL ontologies), and allows for a wider range of heuristic optimisations.

2 Tableaux Reasoning and Architecture

Most modern DL systems are based on tableaux algorithms. Such algorithms were first introduced by Schmidt-Schauß and Smolka [15], and subsequently extended to deal with ever more expressive logics [1]. Many systems now implement the $\mathcal{SHIQ}$ DL, a tableaux algorithm for which was first presented in [11]; this logic is very expressive, and corresponds closely to the OWL ontology language. In spite of the high worst case complexity of the satisfiability/subsumption problem for this logic (ExpTime-complete), highly optimised implementations have been shown to work well in many realistic (ontology) applications [7].

Tableaux algorithms work by trying to construct a labelled graph (usually a tree) representation of a model of a concept, the structure of which is determined by syntactic decomposition of the concept. The decomposition and construction is usually carried

[1] FaCT++ is avaliable at `http://owl.man.ac.uk/factplusplus`.

out by applying so called tableaux expansion rules to the concepts in node labels, with one rule being defined for each of the syntactic constructs in the language (with the exception of negation, which is pushed inwards, using re-writings such as de Morgan's laws, until it applies only to atomic concepts).

Most implementations use a top-down construction based on the so-called trace technique [5]. The idea of this technique is to apply *generating rules* (rules which add new leaf nodes to the tree) with the lowest priority (i.e., only apply these rules when no other rules are applicable), and to discard fully expanded sub-trees, so that only a single "trace" (i.e., a branch of the tree) is kept in memory at any one time. This allows (for PSpace logics) tree structures that may be exponential in size (with respect to the size of the input concept) to be delineated using only polynomial space. For the ExpTime logics implemented in modern systems, however, guaranteeing polynomial space usage is no longer an option. Moreover, for logics that support inverse roles (such as $\mathcal{SHIQ}$), a strictly top down approach is no longer practical as constraints may be propagated both "up" and "down" the edges in the tree.

In spite of this, top-down construction is still widely used: it has the advantage of being very simple and easy to implement—a procedure that exhaustively expands a node label can be applied to the current node and then, recursively, to each of its successors. It does, however, have some serious drawbacks. In the first place, for logics with inverse roles, a strictly top-down construction would break down as it relies on the fact that rules only ever add concepts to the label of the node to which they are applied or to the label of one of its successor nodes. In the presence of inverse roles, expansion rules may also add concepts to the labels of predecessor nodes, which could then require further expansion. Moreover, discarding fully expanded sub-trees may no longer be possible, as the expansion of a concept added to the label of a predecessor may cause concepts to be added to the label of a sibling node that had previously been fully expanded.

In the second place, the top down method forces non-deterministic rules (e.g., the rule for expanding disjunctions) to be applied with a higher priority than generating rules. As the size of the search space caused by non-deterministic rule expansions is, in practice, by far the most serious problem for tableaux based systems [6], it may be advantageous to apply non-deterministic rules with the lowest priority [4]. In fact, top-down implementations typically apply non-deterministic rules with a priority that is lower than that of all of the other rules *except* the generating rules [9].

ToDo List Architecture The FaCT++ system was designed with the intention of implementing DLs that include inverse roles, and of investigating new optimisation techniques, including new ordering heuristics. Currently, FaCT++ implements $\mathcal{SHIF}$, a slightly less expressive variant of $\mathcal{SHIQ}$ where the values in cardinality restrictions can only be zero or one.[2]

Instead of the top-down approach, FaCT++ uses a *ToDo list* to control the application of the expansion rules. The basic idea behind this approach is that rules may become applicable whenever a concept is added to a node label. When this happens, a note of the node/concept pair is added to the ToDo list. The ToDo list sorts all entries according to some order, and gives access to the "first" element in the list. A given tableaux algorithm takes an entry from the ToDo list and processes it according to the expansion rule(s) relevant to the entry (if any). During the expansion process, new con-

[2] $\mathcal{SHIF}$ corresponds to the OWL-Lite ontology language [10].

cepts may be added to node labels, and hence entries may be added to the ToDo list. The process continues until either a clash occurs or the ToDo list become empty.

This architecture has a number of advantages when compared to the top-down approach. Firstly, it is applicable to a much wider range of logics, including the expressive logics implemented in modern systems, because it makes no assumptions about the structure of the graph (in particular, whether tree shaped or not), or the order in which the graph will be constructed. Secondly, it allows for the use of more powerful heuristics that try to improve typical case performance by varying the global order in which different syntactic structures are decomposed; in a top-down construction, such heuristics can only operate on a local region of the graph—typically a single node.

In FaCT++ the ToDo list architecture is implemented as a set of queues (FIFO buffers). It is possible to set a priority for each rule type, and a separate queue is implemented for each unique priority. Whenever the expansion algorithm asks for a new entry, it is taken from the non-empty queue with the highest priority, and the algorithm terminates when all the queues are empty. This means that if the $\exists$-rule (the generating rule that expands existential restrictions) has a low priority (say 0), and all other rules have the same priority (say 1), then the expansion will be (modulo inverse roles) top-down and breadth first; if stacks (LIFO buffers) were used instead of queues with the same priorities, then the expansion would simulate the standard top-down method.

3 Optimisations

Many of the optimisations used in FaCT++ were taken from the original FaCT. These include lexical normalisation, boolean constraint propagation, dependency-directed backjumping, caching, etc. [1].

The algorithm implemented in FaCT++ is extended to include native support for role range and domain axioms, which leads to improved performance when reasoning with realistic ontologies (which may contain many such axioms). This extension can be also be exploited in order to add a new form of absorption optimisation called *role absorption* [16].

Ordering Heuristics As discussed in Section 4, FaCT++'s ToDo list architecture allows for a wide range of heuristics to be applied to choosing a "good" order in which to apply inference rules (we will call these *rule-ordering* heuristics). Heuristics can also be used to choose, for non-deterministic rules, the order in which to explore the different expansion choices (we will call these *expansion-ordering* heuristics).

FaCT++ includes a range of different expansion-ordering heuristics that can be used to choose the order in which to explore the different expansion choices offered by the non-deterministic $\sqcup$-rule. This ordering can be on the basis of the size, maximum quantifier depth, or frequency of usage of each of the concepts in the disjunction, and the order can be either ascending (smallest size, minimum depth and lowest frequency first) or descending. In order to avoid the cost of repeatedly computing such values, FaCT++ gathers all the relevant statistics for each concept as the knowledge base is loaded, and caches them for later use.

FaCT++ separates the classification process into satisfiability testing (SAT) and subsumption testing (SUB) phases; the results from the SAT phase are cached and used to speed up subsequent tests via a standard "model-merging" optimisation [9]. It is possible to set different optimisation options for each phase of the reasoning process.

4 Empirical Evaluation

We have used several real-life ontologies, as well as artificial test suites, to evaluate different optimisations, and compare FaCT++'s performance with that of other state-

of-the-art DL reasoners. Due to space restrictions, the results of only a small subset of these tests can be presented here.

Ordering tests These tests illustrate how different expansion ordering heuristics can influence the reasoner's performance. Several realistic KBs were used in this test: Wine-Food is a sample ontology that makes up part of the OWL test suite[3] [2]; DOLCE is a foundational (top-level) ontology, developed in the WonderWeb project [3]; and GALEN is the anatomical part of the well-known medical terminology ontology [14]. The tests measured the time taken (in CPU seconds) to classify the ontologies.

Table 1. Ontology classification times (seconds) for ToDo (left) and OR (right) orderings

KB	DOLCE		WineFood		GALEN	
	SAT	SUB	SAT	SUB	SAT	SUB
a	0.74	0.74	0.22	2.44	99.44	1678.11
aO	0.64	0.68	**0.14**	**1.64**	29.80	569, 64
aEO	**0.58**	**0.57**	0.15	1.67	**9.88**	**173.79**
aE	0.60	0.58	0.27	2.87	13.35	205.32
aOE	0.61	0.59	0.27	2.93	13.22	201.40

SAT	Sa	Da	Fa	Sd	Dd	Fd
SUB	0.26	0.29	0.19	0.13	0.13	0.20
Sa	3.15	3.57	3.27	3.21	3.21	3.68
Da	3.54	3.57	3.44	3.20	3.40	3.47
Fa	3.67	3.57	2.32	2.12	2.41	2.35
Sd	1.77	1.80	1.71	1.80	1.80	1.83
Dd	1.69	1.77	1.87	**1.66**	1.78	1.78
Fd	2.30	2.26	2.75	3.14	3.54	2.76

In the rule-ordering tests (left-hand side of Table 1), each ordering strategy is shown as a sequence of letters specifying the priorities (highest first) of the different rule types, where "O" refers to the $\sqcup$-rule, "E" to the $\exists$-rule, and "a" to any other rule type. E.g., "aO" describes the strategy where the $\sqcup$-rule has the lowest priority, and all other rules have an equal higher priority.

In most cases the best result is given by the "aEO" strategy (i.e., by assigning the lowest priority to the $\sqcup$-rule and the next lowest priority to the $\exists$-rule), and even when "aEO" is not the best strategy, the difference between it and the best strategy is very small. Note, that such a strategy is impossible to implement in a top-down approach, because it is necessary to give the lowest priority to the $\exists$-rule.

Table 1 presents the results of expansion-ordering tests using the WineFood ontology. Each strategy is denoted by two letters, the first of which indicates whether the ordering is based on concept size ("S"), maximum depth ("D") or frequency of usage ("F"), and the second of which indicates ascending ("a") or descending ("d") order. When using the best strategy ("Sd" for SAT and "Dd" for SUB), classification takes less than half the time taken when using the worst strategy.

InstanceStore query answering The instanceStore (iS) is a technique used to deal with ontologies containing a very large number of individuals. It combines terminological reasoning with database approaches to answer a limited form of instance-retrieval query against such ontologies. A more detailed description of the iS can be found in [8].

Here we compare the performance of FaCT and FaCT++ in iS query answering. The test used the 40,000 concept Gene Ontology, with roughly 500,000 individuals mined from the GO database [8]. In some cases, query answering may require many subsumption tests to be performed w.r.t. the classified ontology.

Figure 2 shows the results of these tests, including the classification time (in seconds) for the ontology, the answer size (number of instances) and the time taken (in seconds) for each query. It is easy to see that FaCT++ is significantly better than FaCT

[3] This ontology therefore has a much weaker claim to being "realistic".

Table 2. instanceStore classification and query answering times (seconds)

Query	Classification	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Ans. size	—	6,527	0	0	96,105	27	13,449	11,820	12	19	4,563	1
FaCT	247	3.1	37.1	30.1	61.2	0.7	16.9	3.2	87.8	8.8	2.1	2.4
FaCT++	84	3.9	10.6	6.1	38.5	0.5	14.4	1.9	31.4	7.4	1.3	0.1

both in classifying the (large but relatively simply structured) ontology, and in performing large numbers of (relatively easy) subsumption tests.

5 Discussion and Future Directions

We have presented FaCT++, an OWL Lite DL reasoner which uses a new ToDo list architecture and incorporates new optimisations.

Future directions for FaCT++ include both algorithmic and technological improvements. The next version of FaCT++ will support the more expressive $\mathcal{SHOIQ}$ DL needed by the OWL DL ontology language. Some new optimisations, including dynamic backjumping and more elaborate heuristics are also planned. Regarding technological improvements, we plan to add direct support for OWL's RDF syntax, and to implement FaCT++ as a standalone HTTP server.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. CUP, 2003.
2. J. Carroll and J. De Roo. OWL web ontology language test cases. W3C Recommendation, 2004.
3. A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with DOLCE. In *Proc. of EKAW 2002*, 2002.
4. F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for $\mathcal{ALC}$. In *Proc. of KR'96*, pages 304–314, 1996.
5. B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption algorithms for concept description languages. In *Proc. of ECAI'90*, pages 348–353, 1990.
6. I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.
7. I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR'98*, pages 636–647, 1998.
8. I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The instance store: DL reasoning with large numbers of individuals. In *Proc. of DL'2004*, pages 31–40, 2004.
9. I. Horrocks and P. F. Patel-Schneider. Optimizing description logic subsumption. *J. of Logic and Computation*, 9(3):267–293, 1999.
10. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.
11. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, pages 161–180, 1999.
12. H. Knublauch, R. Fergerson, N. Noy, and M. Musen. The protégé OWL plugin: An open development environment for semantic web applications. In *Proc. of ISWC 2004*, 2004.
13. A. Rector. Description logics in medical informatics. In *The Description Logic Handbook*, pages 306–346. CUP, 2003.
14. J. E. Rogers, A. Roberts, W. D. Solomon, E. van der Haring, C. J. Wroe, P. E. Zanstra, and A. L. Rector. GALEN ten years on: Tasks and supporting tools. In *Proc. of MEDINFO2001*, pages 256–260, 2001.
15. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
16. D. Tsarkov and I. Horrocks. Efficient reasoning with range and domain constraints. In *Proc. of DL'2004*, pages 41–50, 2004.

Appendix J:

"Benchmarking DL Reasoners Using Realistic Ontologies" by Zhengxiang Pan, in OWL: Experiences and Directions workshop, Galway, Ireland, November 2005. Enclosed with the full electronic version of this report as the file owled2005-benchmarking.pdf.

Benchmarking DL Reasoners Using Realistic Ontologies

Zhengxiang Pan

Bell Labs Research and Lehigh University
zhp2@lehigh.edu

Abstract. We did a preliminary benchmark on DL reasoners using real world OWL ontologies. First hand experiences on OWL ontologies and reasoning services available to OWL are described and discussed.

1 Introduction

We developed a benchmark to evaluate the performances of state-of-the-art expressive Description Logic reasoners using an empirical approach. The outcomes are expected to illustrate the effectiveness of DL reasoners that vary in optimization techniques and implementation strategies. Some observations on the deployment of OWL language are also made and discussed.

We conducted the benchmark toward three well-known DL reasoners: Racer, FaCT++ and Pellet. The benchmark data consists of more than one hundred most deployed OWL ontologies across many domains, from life science [1] to geographic [2], from food and wines [3] to stock exchange [4].

Because of the heterogeneous interfaces (DIG and HTTP etc.) being used in the different systems, this benchmark should be regarded as more qualitative than quantitative, i.e. precise time counting is of little interests here. Instead, we will try to analyze the detailed outputs and logs of those reasoning systems and extract useful informations.

2 Background

Due to the correspondence between OWL and description logics, reasoning support to OWL applications heavily relies on description logic reasoners. Benchmarks and evaluations on DL reasoners thus become an important task that concerns with OWL.

A number of efforts have been made on benchmarking DL reasoner. Some of them take the road of generating synthetic formulas randomly [5] [6] [7]. Although this approach could lead to a more comprehensive benchmark on reasoners, the parameters of those generators need to be fine tuned by sampling the realistic KBs. In contrast, using the real world KBs as test data would be more feasible and efficient, the results would be more easily interpreted and used by end users. Part of the test in [8] was using realistic TBoxes, but only 8 KBs

were included at that time. An analysis of a DAML+OIL ontology collection in [9] characterized and categorized the real-world ontologies but the depicted benchmark was not implemented and conducted.

3 Experiments

3.1 Target Systems

Three DL reasoners were chosen to run the benchmark: Racer, Pellet and FaCT++. Certainly there are other DL reasoners worth benchmarking but were not tested due to various constraints.

According to [10], Racer implementation employed tableaux calculus for $\mathcal{SHIQ}$ as well as the following optimization techniques: dependency-directed backtracking and DPLL-style semantic branching, transformation of axioms (GCIs), model caching and model merging.

Pellet is claimed to be sound and complete on $\mathcal{SHIN}(\mathcal{D})$ and $\mathcal{SHON}(\mathcal{D})$ [11]. It implements TBox partitioning, absorption and lazy unfolding plus dependency directed backjumping, semantic branching and early blocking strategies. It also supports datatype reasoning and uses some optimizations in Abox query answering.

Not quite similar to FaCT, FaCT++ implemented a new tableaux decision procedure for $\mathcal{SHOIQ}$ [12]. Being under its early stage of development, FaCT++ has very limited user interface and no API is available.

3.2 Test Data

Our test data consists of 135 real world OWL ontologies. They were submitted by the ontology authors and users from different domains.

Our original plan was to use an OWL-aware crawler to crawl OWL ontologies available on the web. However, we did not find such a tool. Hence we turned to finding a large collection of OWL ontologies. Among handful candidates, Schemaweb [13] which is a RDF schemas directory became our choice. A java program was developed then to read and identify those schemas.

We fed 250 urls indexed on Schemaweb into our program and we identified 135 OWL ontologies among them. Totally 5897 classes and 2601 relations were recorded out of these OWL ontologies.

Having their subjects distributed in a broad range of domains, these OWL ontologies also vary in size, constructs being used and complexity. Thus, we argue these 135 OWL ontologies largely represent the current usage of OWL language in practical despite that they are just a small portion of existing OWL documents.

3.3 Experiment Configurations

The experiments were done on a Linux box featuring an Intel(R) Pentium(R) 4 CPU at 2.6GHz and 1 giga bytes main memory.

For each target system, script or special handling program were written to direct the benchmark. No matter how it was being executed, the benchmark is the iteration of the following steps:

1. Clear the memory and cache in the application.
2. Read in the next ontology in the test set.
3. Do classification.

A time limit of one hour (3600 seconds) was set for each ontology, meaning that any processes regarding a particular ontology will be aborted if aggregated CPU time exceeds 3600 seconds.

Racer:

RacerPro 1.8.1 has recently released as a commercial software; however in this paper we used the last free-for-research version Racer 1.7. A racer instruction file was created to run the benchmark. Each ontology corresponds to four commands in that file. First two commands (DELETE-ALL-ABOXES) and (DELETE-ALL-TBOXES) cleaned up the memory. Then (OWL-READ-DOCUMENT "*url*") command asked Racer to read in the specified ontology. At the end, (TBOX-COHERENT-P) and (ABOX-CONSISTENT-P) invoked the classifications in the Reasoner.

Pellet:

We used the Pellet 1.1.0 released on 05/07/2004. A script file was created to manage the benchmark. Each parameterized execution of Pellet would read one ontology and do the classification. Since each ontology was processed by a fresh start of the Pellet, no need to clean the memory and cache in this case.

FaCT++:

As part of the aforementioned limitations, FaCT++ doesn't take OWL documents directly nor any remote files. A utility program digFaCT++ takes local files in DIG [14]. In order to make the benchmark working, we developed a java program to translate the OWL ontologies into DIG format and store them locally. In the benchmark script, each execution of digFaCT++ was supplied two parameters. One is a tell-document in DIG corresponding to one ontology, the other is a simple query file that only ask if TOP is satisfiable. This simple query was used here to invoke the classification in FaCT++.

4 Results

Racer finished the benchmark in about 15 minutes. It successfully made TBox classifications on 108 ontologies, 101 of which were found to be consistent. It also made successful consistency check on ABox for 92 ontologies, 83 of which were found to be consistent. For those aborted tasks and inconsistent ontologies, Racer reported 117 errors, about one third of which is due to the syntax errors or usages beyond the scope of OWL DL.

Pellet had done classifications on 103 ontologies within the time limit. It spent almost 2 hours (6814 seconds) on these ontologies. Interestingly, all these finished ontologies were classified to be consistent. However, Pellet automatically did

somethings more than just classification. It validates the species of the ontologies and tries to repair OWL Full ontologies if they are missing type triples [11]. In our benchmark result, 70 out of 103 ontologies were validated as OWL Full, 23 and 10 for DL and Lite respectively.

Except timed out for 3 ontologies, FaCT++ had done the remaining in nearly 30 seconds. Its log recorded that it only spent 2.6 seconds on classifications of the 121 ontologies, which were all successfully classified. Note the time of parsing and I/O was not included, nor was the time spent on translating OWL into DIG. Nevertheless, this kind of performance was very impressive.

System	Consistent	Inconsistent	Timed out	Aborted
Racer(Tbox)	101	7	0	27
Pellet	103	0	17	15
Fact++	121	0	3	11

Table 1. The Results of Classification: Performed on 135 ontologies

5 Discussion

Here we summarize some interesting observations from our benchmark results. They could potentially give us some helpful hints on ontology authoring as well as the design and implementation of reasoners.

Firstly, the test data and the output of reasoners gave us a good chance to characterize the current usage of OWL language. Based on the result from Pellet, more than 70% of the classified ontologies are OWL Full, more than three quarters of these OWL Full ones can be validated as OWL DL just by adding some statements, like type triples. Racer also found out 22 cases where transitive properties were used in cardinality restrictions, legitimate only in OWL Full.

In addition, we used the WonderWeb validator [15] to validate the species of the rest ontologies. By adding these up we get figure 1, showing that the majority of the test KBs are OWL Full. Note the "unknown" category was for those ontologies that caused errors on the validator.

We assume that only few authors intended to create an ontology in OWL Full, because of the extreme difficulties in finding reasoning support. Thus, an ontology editor with built-in validator and heuristic non-DL finder is highly desired and should be widely adopted. Some efforts has been made toward this direction, such as [16].

Secondly, the performances of the reasoners varied a lot. Although they are not directly comparable due to the different input formats (OWL v.s. DIG) and side-functionalities (species checking etc.), the results implies the effectiveness of some optimization techniques being deployed. Apparently, FaCT++ completed the most testing ontologies using the least time. Within the scope of classification, FaCT++ significantly pushed the baseline of DL reasoners to a new high.

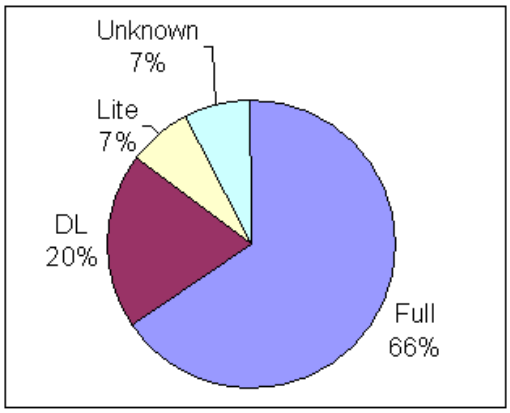

Fig. 1. Percentages of Each Species of OWL in Test Data

However, FaCT++ provides very limited services compare to other rivals. So far it only supports queries through DIG, which is not very expressive in ABox retrieving.

Pellet, on the other hand, done least TBox classifications with most time. Unfortunately, we have no way to figure out how much time was actually spent on those extra functionalities such as species checking. Figure 2 shows that classification time in Pellet increased nearly exponentially, no wonder 17 testing ontologies were timed out. Interestingly, we found out that some of the ontologies that Pellet spent a huge amount of time on (but finished) were the ones timed out or failed by FaCT++. This suggests that to some extent, Pellet is more resilient to non-trivial ontologies.

For most of the testing documents, Racer was not as fast as FaCT++, but it never timed out. This intrigues a dilemma on the implementation strategies: give time or give up? Guaranteed termination is a nice property but sometimes resilience is also desired. One possible solution is to allow users to customize the time-out settings for each execution.

Above all, the performance of a DL reasoner is affected by the following factors:

- The quality of inputs. DL reasoners are not intended to perform on non-DL ontologies. Reasoner should be able to identify its capability on the given knowledge base before long deliberations.
- The optimization techniques. Other experiences [17] show that cyclic axioms, inverse roles and nominals are "killer" constructs for DL reasoners. New optimizations should target these cases.

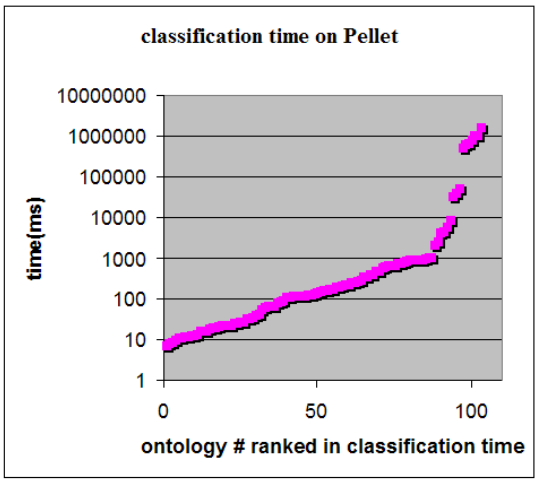

Fig. 2. Performance degradation of Pellet

– The feasibility for customization. Different applications have different constraints or preferences on speed and/or expressiveness. They require different side-functionalities even just for classification.

6 Conclusion

We performed a preliminary benchmark on three state-of-the-art DL reasoners: Racer, Pellet and FaCT++. They vary from each other in many aspects, even in programming languages: Lisp, Java and C++ respectively. Real world OWL ontologies across various domains were used as test KBs. Observations on the characteristics of those OWL ontologies as well as the performances of the reasoners were reported and discussed. We do not intend to use the results as direct reflection of those systems' overall performances, for that this simple benchmark is not systematic enough and only focus on TBox classification at this time.

There are a couple of future directions on this work. Firstly we should identify and study case by case on these non-trivial KBs, i.e. the ones timed out, failed or spent a lot of time. These will intrigue research directions on DL reasoner optimizations. Secondly we need to formalize our benchmark by making the targeting reasoners more comparable, probably wrapping them in the same API. Furthermore, the detailed relationship between optimization techniques and performances should be analyzed.

7 Acknowledgement

This work was carried out while the author was visiting Bell Labs at Murray Hill, New Jersey, USA.

References

1. (http://www.cs.man.ac.uk/ wroec/mygrid/ontology/mygrid.owl)
2. (http://loki.cae.drexel.edu/ wbs/ontology/2003/10/iso-metadata.owl)
3. (http://www.w3.org/TR/owl-guide/wine.rdf)
4. (http://www.daml.org/2001/10/html/nyse-ont)
5. Q. Elhail, M.R., Ycart, B.: Generating random benchmarks for description logics. In: Proc. of DL'98. Volume 11. (1998)
6. Patel-Schneider, P.F., Sebastiani, R.: A new general method to generate random modal formulae for testing decision procedures. J. Artif. Intell. Res. (JAIR) **18** (2003) 351–389
7. Hladik, J.: A generator for description logic formulas. In Horrocks, I., Sattler, U., Wolter, F., eds.: Proceedings of DL 2005, CEUR-WS (2005) Available from `ceur-ws.org`.
8. Horrocks, I., Patel-Schneider, P.F.: Dl systems comparison (summary relation). In: Description Logics. (1998)
9. Tempich, C., Volz, R.: Towards a benchmark for semantic web reasoners - an analysis of the daml ontology library. In: EON. (2003)
10. Haarslev, V., Möller, R.: Racer system description. In Goré, R., Leitsch, A., Nipkow, T., eds.: International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy, Springer-Verlag (2001) 701–705
11. Parsia, B., Sirin, E.: Pellet: An owl dl reasoner. In: Proc. International Semantic Web Conference. (2004)
12. Horrocks, I., Sattler, U.: A tableaux decision procedure for $\mathcal{SHOIQ}$. In: Proceedings of Nineteenth International Joint Conference on Artificial Intelligence. (2005)
13. (http://www.schemaweb.info)
14. Bechhofer, S.: The DIG Descriprion Logic Interface: DIG/1.1. (2003)
15. (http://phoebus.cs.man.ac.uk:9999/OWL/Validator)
16. Bechhofer, S., Volz, R.: Patching syntax in owl ontologies. In: Proceedings of the Third International Semantic Web Conference. (2003)
17. Haarslev, V., Möller, R., Wessel, M.: Description logic inference technology: Lessions learned in the trenches. In Horrocks, I., Sattler, U., Wolter, F., eds.: Proc. International Workshop on Description Logics. (2005)

Appendix K:

"Automated Benchmarking of Description Logic Reasoners" by Tom Gardiner, Ian Horrocks, and Dmitry Tsarkov, in Proceedings of the 2006 Description Logic Workshop (DL-2006), June 2006. Enclosed with the full electronic version of this report as the file dl2006-benchmarking.pdf.

Automated Benchmarking of Description Logic Reasoners

Tom Gardiner, Ian Horrocks, Dmitry Tsarkov
University of Manchester
Manchester, UK
{gardint3|horrocks|tsarkov}@cs.man.ac.uk

March 10, 2006

1 Introduction

Reasoning for expressive DLs implemented in state-of-the-art systems has high worst case complexity. The hope/claim is, however, that these systems perform well in "realistic" applications. In practice, this means in ontology applications. To check the validity of this claim it is necessary to test the performance of these systems with (the widest possible range of) ontologies derived from applications.

In addition, testing is useful in order to check the correctness of implementations. In small examples, it may be easy to check the correctness of a system's reasoning. However, for typical real-world examples, manual checking is not feasible. In these instances, the best (perhaps the only) way to check correctness is often by checking for consistency with the reasoning of other existing systems.

Real-world ontologies vary considerably in their size and expressivity. While they are all valuable test cases, it is still important to understand each ontology's properties in order to provide efficient and relevant testing.

System developers find this particularly useful, as it helps them to identify weaknesses in their systems and to devise and test new optimisations. Finally, testing is also useful for the developers and users of applications as they can use benchmarking results to determine if (the performance of) a DL reasoner is likely to satisfy their requirements, and if so which reasoner is likely to perform best in their application.

2 Background and Related Work

For the above mentioned reasons, there is extensive existing work on benchmarking DL (as well as modal logic) reasoners. E.g., TANCS comparisons and bench-

mark suites [10], DL comparisons and benchmark suite [1], work on M-SPASS [8], work on FaCT and DLP [7, 6], the OWL benchmark suite and test results, and various test results from papers describing systems such as FaCT++ (`http://owl.man.ac.uk/factplusplus`), Pellet (`http://www.mindswap.org/2003/pellet/`), Racer [5], KAON2 (`http://kaon2.semanticweb.org/`), Vampire [12], etc.

Due to the fact that relatively few (large and/or interesting) ontologies were available, earlier tests often used artificially generated test data. The Lehigh University Benchmark [4], for example, used a synthetic ontology and randomly generated data to test the capabilities of knowledge base systems using specific weightings to compare systems on characteristics of interest. Results from such tests are, however, of doubtful relevance when gauging performance on ontologies. The popularity of OWL has meant that many more ontologies are now available, and recent benchmarking work has focused on testing performance with such ontologies.

One such example [11] involved benchmarking a number of reasoners against a broad range of realistic ontologies. However, not all reasoners used in that comparison supports OWL as an input language, so quantitative comparison of performance would have been difficult/un-justified. From the other hand, the DIG interface [2] is recognised as a preferred choice by application developers and thus is implemented into a wide range of DL Reasoners.

Our work builds on these earlier efforts, taking advantage of the DIG standard to provide a generic benchmarking suite that allows the automatic quantitative testing and comparison of DL Reasoners on real-world examples with relevant properties. We aim to make the testing process as autonomous as possible, taking care, for example, of (re)starting and stopping reasoners as necessary, and the analysis of results be as flexible as possible, by allowing for arbitrary SQL queries against the collected data. We also aim to provide, as a publicly available resource, a library of test ontologies where each ontology has been checked for expressivity and syntactic conformance, translated into DIG syntax (which is much easier to work with that OWL's RDF/XML syntax), and includes (where possible) results (such as the concept hierarchy) that can be used for testing the correctness of reasoning systems.

3 Methodology

The system has two main functions. The first is to process ontologies and add them to the library, and the second is to benchmark one or more reasoners using the ontology library.

When processing ontologies, the system takes as input a list of OWL-ontology URI's. Before they can be used in testing, some preprocessing of these ontologies is required. The process involves generating valuable meta-data about each

ontology, as well as converting each of the OWL-ontologies to DIG.

The meta-data is generated by code written for SWOOP [9], and provides the details of the expressivity (i.e. the constructs present in the ontology) together with the number of classes, object properties, data properties, individuals, class axioms, property axioms and individual axioms present. The OWL-to-DIG conversion uses the OWL-API (`http://owlapi.sourceforge.net`). This process is far from trivial as OWL's RDF syntax is extremely complex, and it is easy to (inadvertently) cause ontologies to be outside of OWL DL, e.g., by simply forgetting to explicitly type every object. Moreover, the DIG interface supports only the most basic of data types, such as Strings and Integers. The result is that many of the available OWL Ontologies we found could not be successfully converted to DIG.

Local copies are stored of both the OWL Ontology and the DIG version. This is not only for efficiency during the testing, but also to ensure consistency (as online ontologies rarely remain static). Moreover, this allows us to fix trivial errors in the OWL ontologies so that they can be used for testing purposes. The locations of these files, together with their properties/meta-data, are stored as database entries for easy access and manipulation.

The main function of the benchmark suite itself is timing the classification of each ontology by each Reasoner. CPU time is the measure used as it eliminates external factors such as background threads, and to promote fairness, each Reasoner is terminated and then restarted for every test.

A problem with trying to compare different Reasoners is that they may perform tasks in different ways. For example, they may vary in the way in which they perform each part of the reasoning: some may take an "eager" approach, fully classifying the whole ontology and caching the results as soon as it is received; others may take a "lazy" approach, only performing reasoning tasks as required in order to answer queries. To try to get around this problem, we use a five step test that forces every reasoners to fully classify each ontology. The tests are as follows:

1. TELLS the reasoner the full ontology
2. ASK for all the concepts in the ontology
3. ASK for the satisfiability of the TOP concept
4. ASK for the satisfiability of all the concepts in the ontology
5. ASK for the ontology taxonomy (parents and children of all concepts)

Each of these individual tests are then timed, providing interesting information about when different reasoners do most their work. It is, however, the total time for this complete classification that we are most interested in.

Each classification will then end in one of three ways. It will either complete successfully, fail due to lack of time or fail for some other reasons. The latter may include fail due to lack of run-time memory, fail because the reasoner could not parse the ontology successfully, etc.

The benchmark suite is fully automatic, dealing with most errors autonomously,

meaning that the testing can be left to run over-night or over a week-end (which may be necessary when using a large time-out). All data is recorded in a MySQL database, making it easy to the user to view and analyse the data in a variety of ways.

As discussed in Section 1, in order to get a clearer indication of how DL Reasoners perform in the real world, we aim to build a large library of OWL ontologies from those that are publicly available. Currently, our library contains a little over 300 OWL-RDF Ontologies, but only 172 of these could successfully be converted to DIG. This has, however, provided us with a total of just under 72,000 classes and over 30,000 individuals in a DIG format. Only 18% of the ontologies were at least ALC, which suggests that the majority of real-world ontologies aren't in fact very complex, but it also means we have a comfortable number of "interesting" examples too.

4 Testing

Our system is currently fully automatic and runs the classification tests successfully through our whole library. It does not, however, at this stage verify the correctness of each Reasoner's answers to the queries and how they compare to the answers given by other Reasoners. This means that our measure of success is, for now, merely an indication that the Reasoner received and parsed the DIG successfully and returned a valid DIG response. This is generally a good indication, but should only be considered a preliminary result.

We have performed some tests on our system, as it stands, and we provide here some examples of the kinds of information that our system can produce.

FaCT++ v1.1.2, KAON2, Pellet v2.2 and RacerPro v1.9 are four of the most widely used OWL/DIG reasoners, and we therefore decided to use these to test the current capabilities of our system. The tests were performed using an Intel Pentium M Processor 1.60 GHz and 1Gb of Main Memory on Windows XP. The time-out period was set to one hour (in real time). Pellet and KAON2 are java applications, and for these tests were run with a maximum heap space of 200Mb. RacerPro and FaCT++ were left to run on their default settings.

Table 1 shows how the individual Reasoners performed firstly on all our ontologies and then on Ontologies which have particular characteristics. Finally, it shows their performance on OWL-Lite ontologies, which includes all those with expressivity up to SHIF.

In order to determine which were the most "challenging" ontologies (w.r.t. reasoning), we tried to order ontologies according to the difficulty of reasoning with them. To do this, we used all the ontologies that were successfully classified by at least two Reasoners and then ordered these by their average classification time. Figure 1 shows the amount of time each Reasoner took to classify the 10 most challenging ontologies according to this measure (where negative time

Type	Status	FaCT++	KAON2	Pellet	RacerPro
All	Success	127	59	167	128
All	Failed	40	113	4	44
All	TimedOut	5	0	1	0
Nominals	Success	10	3	9	12
Nominals	Failed	2	9	3	0
Nominals	TimedOut	0	0	0	0
TransRoles	Success	17	10	20	16
TransRoles	Failed	3	12	2	6
TransRoles	TimedOut	2	2	0	0
Datatypes	Success	90	21	123	87
Datatypes	Failed	35	106	4	40
Datatypes	TimedOut	2	0	0	0
OWL-Lite	Success	32	36	39	39
OWL-Lite	Failed	5	4	0	1
OWL-Lite	TimedOut	3	0	1	0

Table 1: Sample of Overall Performance

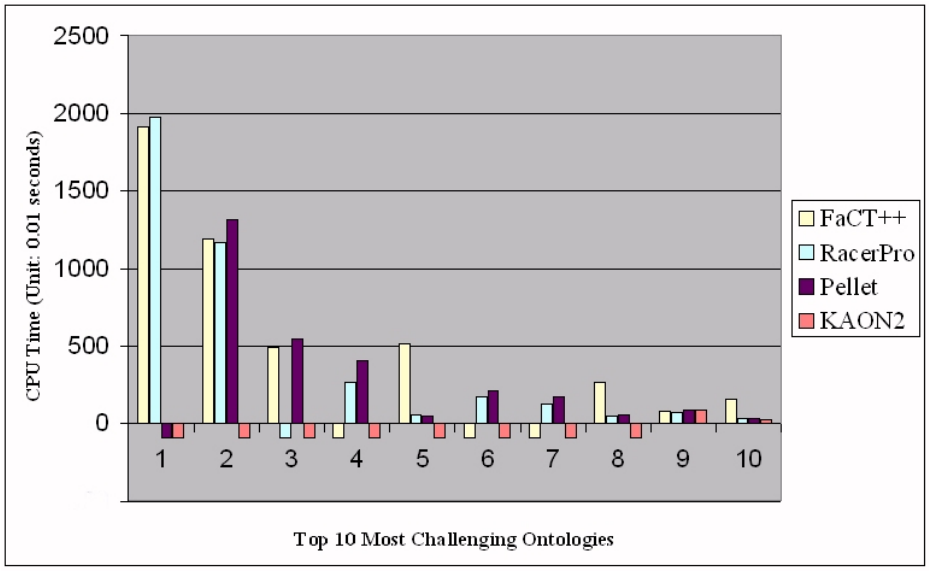

Figure 1: Comparison of Reasoners on the Top 10 Most Challenging Ontologies

Ontology	Expressivity	nClass	nIndiv	URL
1	DLLite	27652	0	http://...logy/nciOncology.owl
2	ALR+	20526	0	http://archive.godatabase.org/
3	SHIN	395	0	http://...gies/tambis-full.owl
4	RDFS(DL)	1108	3635	http://...world-fact-book.daml
5	RDFS(DL)	4	1900	http://...nt/AirportCodes.daml
6	SHF	3097	0	http://...ibrary/not-galen.owl
7	SHF	2749	0	http://...Ontologies/galen.owl
8	ALR+HI(D)	5	2744	http://...ogicUnits/2003/09/hu
9	RDFS(DL)	1514	0	http://...logy/data_center.owl
10	SI(D)	35	2765	http://...w/HydrologicUnits/hu

Table 2: Properties of Top 10 Most Time-consuming Ontologies

Reasoner	Tells	ConceptList	SatOfTop	SatOfClasses	Hierarchy
FaCT++	6%	16%	16%	11%	51%
KAON2	13%	9%	8%	26%	44%
Pellet	24%	14%	12%	18%	33%
RacerPro	33%	7%	11%	13%	36%

Table 3: Average Division of Task Time

represents a failure to classify). Please note that the times are given in CPU time and are considerably lower than the real time taken for classification (as a Windows machine will typically have hundreds of threads running concurrently). Table 2 then shows some of the interesting information that is available on these "Top 10" Ontologies.

This table is useful in helping us understand what makes these particular Ontologies so time-consuming to reason over. In the case of the NCI and Gene Ontology's (1st and 2nd), it can be clearly seen that it is their shear size that provides the challenge. The 4th and 5th (world-fact-book and AirportCodes) make up for their number of classes with an extensive array of individuals. Whereas Tambis (3rd) simply uses some very complicated constructs.

Our final table, Table 3, shows the average proportion of each classification test that each Reasoner spent on the separate tasks. This shows, for example, that RacerPro performs a lot of caching on receiving the Ontology (TELLS), while FaCT++ does nothing until the first ASK query.

5 Discussion

As we mentioned in the introduction, testing is useful for reasoner and tool developers as well as for users. Building on existing work, we have developed

a system for testing reasoners with available ontologies. The benefits of our approach include autonomous testing, flexible analysis of results and the development of a test library that should be a valuable resource for both the DL and ontology community. We will continue to extend the library, and will add classification results from tested reasoners so that correctness testing can also be performed.

While there are an increasingly large array of OWL-Ontologies available for public use, other Ontology formats (e.g. OBO: the Open Biomedical Ontologies, `http://obo.sourceforge.net`) are still widely in use and would make for valuable test examples. It is also the case, as describe in [3], that a large proportion of the available OWL-Full Ontologies, could in fact be validated as OWL-DL, just by adding a few extra clarifying statements. This means that of the 162 Ontologies that we had to throw away, many could be useful examples with a little work. In the future we hope to use these observations, together with any external contributions, to considerably increase the size of our ontology library.

The results produced by our tests provide an interesting insight into the variety and depth of information that can be extracted from such testing/benchmarking. However, for the system and it's results to become a valuable resource, we need to test their correctness. We are currently assuming that both the OWL-to-DIG conversions and the Reasoner's responses are all valid and correct.

With regard to the OWL-API's conversions, this was the utility built alongside the original DIG specification. We therefore argue that this is the best conversion available and that our assumption is justified.

Regarding the responses, as discussed earlier, they can be almost impossible to check for correctness. Our best option is therefore to analyse the difference in responses received from different reasoners, and this route is thus one we aim to explore further. It will be interesting to see if reasoners (that should, in theory, all produce the same inferences to the same problems) will actually agree on the test ontologies.

So far we have focused on testing Tbox reasoning (classification). Although the use of nominals in SHOIQ blurs the separation between Tbox and Abox, it would still be useful to explicitly test Abox reasoning, e.g., by asking for the instances of some query class. This functionality will be added in a future version of the system.

Apart from the future work described above, there are a number of extensions to our benchmarking system that would enhance its utility. Allowing users to define their own customised test, rather than the 5 step classification we are using, is one example that would allow Reasoner developers to test specific optimisations and implementations as they are developed. Other relevant tests would include testing how multiple concurrent tests on a Reasoner affects performance, as well as simply not restarting a Reasoner between tests.

We intend for the whole system, including the ontology library, to be available for open-source use in the near future.

References

[1] P. Balsiger and A. Heuerding. Comparison of theorem provers for modal logics. In *Proceedings of Tableaux'98*, May 1998.

[2] Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG description logic interface. In *Proceedings of DL2003 International Workshop on Description Logics*, September 2003.

[3] Sean Bechhofer and Raphael Volz. Patching syntax in OWL ontologies. In *Proceedings of 3rd International Semantic Web Conference, ISWC*, 2004.

[4] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.

[5] V. Haarslev and R. Möller. RACER system description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy*, pages 701–705. Springer-Verlag, 2001.

[6] I. Horrocks. Benchmark analysis with FaCT. In *Proc. of TABLEAUX 2000*, number 1847, pages 62–66. Springer-Verlag, 2000.

[7] I. Horrocks and P. F. Patel-Schneider. FaCT and DLP. In *Proc. of TABLEAUX 98*, pages 27–30, 1998.

[8] U. Hustadt and R. A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. 2000.

[9] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca-Grau, and James Hendler. Swoop: A 'web' ontology editing browser. *Journal of Web Semantics*, 4(2), 2005.

[10] Fabio Massacci and Francesco M. Donini. Design and results of TANCS-00. volume 1847, 2000.

[11] Zhengxiang Pan. Benchmarking DL reasoners using realistic ontologies. In *Proc. of the OWL: Experiences and Directions Workshop*, 2005.

[12] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

Appendix L:

"The Even More Irresistible SROIQ" by Ian Horrocks, Oliver Kutz, and Ulrike Sattler, in Proceedings of the Tenth International Conference on Knowledge Representation and Reasoning, June 2006. Enclosed with the full electronic version of this report as the file *kr2006-sroiq.pdf*.

The Even More Irresistible $\mathcal{SROIQ}$

Ian Horrocks and **Oliver Kutz** and **Ulrike Sattler**
School of Computer Science, The University of Manchester
Kilburn Building, Oxford Road, Manchester M13 9PL, UK
{Horrocks, Kutz, Sattler}@cs.man.ac.uk

Abstract

We describe an extension of the description logic underlying OWL-DL, $\mathcal{SHOIN}$, with a number of expressive means that we believe will make it more useful in practice. Roughly speaking, we extend $\mathcal{SHOIN}$ with all expressive means that were suggested to us by ontology developers as useful additions to OWL-DL, and which, additionally, do not affect its decidability and practicability. We consider complex role inclusion axioms of the form $R \circ S \dot{\sqsubseteq} R$ or $S \circ R \dot{\sqsubseteq} R$ to express propagation of one property along another one, which have proven useful in medical terminologies. Furthermore, we extend $\mathcal{SHOIN}$ with reflexive, antisymmetric, and irreflexive roles, disjoint roles, a universal role, and constructs $\exists R.\mathsf{Self}$, allowing, for instance, the definition of concepts such as a "narcist". Finally, we consider negated role assertions in Aboxes and qualified number restrictions. The resulting logic is called $\mathcal{SROIQ}$.

We present a rather elegant tableau-based reasoning algorithm: it combines the use of automata to keep track of universal value restrictions with the techniques developed for $\mathcal{SHOIQ}$. The logic $\mathcal{SROIQ}$ has been adopted as the logical basis for the next iteration of OWL, OWL 1.1.

Introduction

We describe an extension, called $\mathcal{SROIQ}$, of the description logics (DLs) $\mathcal{SHOIN}$ (Horrocks, Sattler, & Tobies, 1999) underlying OWL-DL (Horrocks, Patel-Schneider, & van Harmelen, 2003)[1] and $\mathcal{RIQ}$ (Horrocks & Sattler, 2004). $\mathcal{SHOIN}$ provides most expressive means that one could reasonably expect from the description-logical basis of an ontology language, and was designed to constitute a good compromise between expressive power and computational complexity/practicability of reasoning. It lacks, however, e.g. qualified number restrictions which are present in the DL considered here since they are required in various applications (Wolstencroft *et al.*, 2005) and do not pose problems (Horrocks & Sattler, 2005). That is, we extend $\mathcal{SHOIQ}$—which is $\mathcal{SHOIN}$ with qualified number restrictions—and extend the work begun in Horrocks, Kutz, & Sattler (2005).

Since OWL-DL is becoming more widely used, it turns out that it lacks a number of expressive means which—when considered carefully—can be added without causing too much difficulties for automated reasoning. We extend $\mathcal{SHOIQ}$ with these expressive means and, although they are not completely independent in that some of them can be expressed using others, first present them together with some examples. Recall that, in $\mathcal{SHOIQ}$, we can already state that a role is transitive or the subrole or the inverse of another one (and therefore also that it is symmetric).

In addition, $\mathcal{SROIQ}$ allows for the following:

1. *disjoint roles*. Most DLs can be said to be "unbalanced" since they allow to express disjointness on concepts but not on roles, despite the fact that role disjointness is quite natural and can generate new subsumptions or inconsistencies in the presence of role hierarchies and number restrictions. E.g., the roles `sister` and `mother` should be declared as being disjoint.
2. *reflexive, irreflexive, and antisymmetric roles*. These features are of minor interest when considering only TBoxes not using nominals, yet they add some useful constraints if we also refer to individuals, either by using nominals or ABoxes, especially in the presence of number restrictions. E.g., the roles `knows`, `hasSibling`, and `properPartOf`, should be declared as, respectively, reflexive, irreflexive, and antisymmetric.
3. *negated role assertions*. Most Abox formalisms only allow for positive role assertions (with few exceptions (Areces *et al.*, 2003; Baader *et al.*, 2005)), whereas $\mathcal{SROIQ}$ also allows for statements like $(\mathtt{John}, \mathtt{Mary}) : \neg\mathtt{likes}$. In the presence of complex role inclusions, negated role assertions can be quite useful and, like disjoint roles, they overcome a certain asymmetry in expressivity.
4. $\mathcal{SROIQ}$ provides complex role inclusion axioms of the form $R \circ S \dot{\sqsubseteq} R$ and $S \circ R \dot{\sqsubseteq} R$ that were first introduced in $\mathcal{RIQ}$. For example, w.r.t. the axiom $\mathtt{owns} \circ \mathtt{hasPart} \dot{\sqsubseteq} \mathtt{owns}$, and the fact that each car contains an engine $\mathtt{Car} \dot{\sqsubseteq} \exists \mathtt{hasPart}.\mathtt{Engine}$, an owner of a car is also an owner of an engine, i.e., the following subsumption holds: $\exists \mathtt{owns}.\mathtt{Car} \sqsubseteq \exists \mathtt{owns}.\mathtt{Engine}$.

[1] OWL also includes *datatypes*, a simple form of *concrete domain* (Baader & Hanschke, 1991). These can, however, be treated exactly as in $\mathcal{SHOQ}(\mathbf{D})$/$\mathcal{SHOQ}(\mathbf{D}_n)$ (Horrocks & Sattler, 2001; Pan & Horrocks, 2003), so we will not complicate our presentation by considering them here.

5. $\mathcal{SROIQ}$ provides the *universal role* U. Together with nominals (which are also provided by $\mathcal{SHOIQ}$), this role is a prominent feature of hybrid logics (Blackburn & Seligman, 1995). Nominals can be viewed as a powerful generalisation of *ABox individuals* (Schaerf, 1994; Horrocks & Sattler, 2001), and they occur naturally in ontologies, e.g., when describing a class such as EUCountries by enumerating its members.
6. Finally, $\mathcal{SROIQ}$ allows for concepts of the form $\exists R.\mathsf{Self}$ which can be used to express "local reflexivity" of a role R, e.g., to define the concept "narcist" as $\exists \texttt{likes}.\mathsf{Self}$.

Besides a Tbox and an Abox, $\mathcal{SROIQ}$ provides a so-called *Rbox* to gather all statements concerning roles.

$\mathcal{SROIQ}$ is designed to be of similar practicability as $\mathcal{SHOIQ}$. The tableau algorithm for $\mathcal{SROIQ}$ presented here is essentially a combination of the algorithms for $\mathcal{RIQ}$ and $\mathcal{SHOIQ}$. In particular, it employs the same technique using finite automata as in Horrocks & Sattler (2004) to handle role inclusions $R \circ S \dot{\sqsubseteq} R$ and $S \circ R \dot{\sqsubseteq} R$. Even though the additional expressive means require certain adjustments, these adjustments do not add new sources of nondeterminism and, subject to empirical verification, are believed to be "harmless" in the sense of not significantly degrading typical performance as compared with the $\mathcal{SHOIQ}$ algorithm. Moreover, the algorithm for $\mathcal{SROIQ}$ has, similar to the one for $\mathcal{SHOIQ}$, excellent "pay as you go" characteristics. For instance, in case only expressive means of $\mathcal{SHIQ}$ are used, the new algorithm will behave just like the algorithm for $\mathcal{SHIQ}$.

We believe that the combination of properties described above makes $\mathcal{SROIQ}$ a very useful basis for future extensions of OWL DL.

The Logic $\mathcal{SROIQ}$

In this section, we introduce the DL $\mathcal{SROIQ}$. This includes the definition of syntax, semantics, and inference problems.

Roles, Role Hierarchies, and Role Assertions

Definition 1 *Let* $\mathbf{C}$ *be a set of* ***concept names*** *including a subset* $\mathbf{N}$ *of* ***nominals****,* $\mathbf{R}$ *a set of* ***role names*** *including the universal role* U*, and* $\mathbf{I} = \{a, b, c \ldots\}$ *a set of* ***individual names****. The set of* ***roles*** *is* $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$*, where a role* R^- *is called the* ***inverse role*** *of* R*.*

As usual, an ***interpretation*** $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ *consists of a set* $\Delta^\mathcal{I}$*, called the* ***domain*** *of* $\mathcal{I}$*, and a* ***valuation*** $\cdot^\mathcal{I}$ *which associates, with each role name* R*, a binary relation* $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$*, with the universal role* U *the universal relation* $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$*, with each concept name* C *a subset* $C^\mathcal{I} \subseteq \Delta^\mathcal{I}$*, where* $C^\mathcal{I}$ *is a singleton set if* $C \in \mathbf{N}$*, and, with each individual name* a*, an element* $a^\mathcal{I} \in \Delta^\mathcal{I}$*. Inverse roles are interpreted as usual, i.e., for each role* $R \in \mathbf{R}$*, we have*

$$(R^-)^\mathcal{I} = \{\langle y, x\rangle \mid \langle x, y\rangle \in R^\mathcal{I}\}.$$

Obviously, $(U^-)^\mathcal{I} = (U)^\mathcal{I}$. Note that, unlike in the cases of $\mathcal{SHIQ}$ or $\mathcal{SHOIQ}$, we did not introduce *transitive role names*. This is because, as will become apparent below, role box assertions can be used to force roles to be transitive.

To avoid considering roles such as R^{--}, we define a function Inv on roles such that $\mathsf{Inv}(R) = R^-$ if $R \in \mathbf{R}$ is a role name, and $\mathsf{Inv}(R) = S \in \mathbf{R}$ if $R = S^-$.

Since we will often work with finite strings of roles it is convenient to extend both $\cdot^\mathcal{I}$ and $\mathsf{Inv}(\cdot)$ to such strings: if $w = R_1 \ldots R_n$ is a string of roles R_i $(1 \leq i \leq n)$, we set $\mathsf{Inv}(w) = \mathsf{Inv}(R_n) \ldots \mathsf{Inv}(R_1)$ and $w^\mathcal{I} = R_1^\mathcal{I} \circ \ldots \circ R_n^\mathcal{I}$, where $\circ$ denotes composition of binary relations.

A *role box* $\mathcal{R}$ consists of two components. The first component is a *role hierarchy* $\mathcal{R}_h$ which consists of (generalised) *role inclusion axioms*. The second component is a set $\mathcal{R}_a$ of *role assertions* stating, for instance, that a role R must be interpreted as an irreflexive relation.

We start with the definition of a (regular) role hierarchy whose definition involves a certain ordering on roles, called *regular*. A strict partial order $\prec$ on a set A is an irreflexive and transitive relation on A. A strict partial order $\prec$ on the set of roles $R \cup \{R^- \mid R \in \mathbf{R}\}$ is called a **regular order** if $\prec$ satisfies, additionally, $S \prec R \iff S^- \prec R$, for all roles R and S. Note, in particular, that the irreflexivity of $\prec$ ensures that neither $S^- \prec S$ nor $S \prec S^-$ hold.

Definition 2 ((Regular) Role Inclusion Axioms) *Let* $\prec$ *be a regular order on roles. A* ***role inclusion axiom*** *(RIA for short) is an expression of the form* $w \dot{\sqsubseteq} R$*, where* w *is a finite string of roles not including the universal role* U*, and* $R \neq U$ *is a role name. A* ***role hierarchy*** $\mathcal{R}_h$ *is a finite set of RIAs. An interpretation* $\mathcal{I}$ ***satisfies*** *a role inclusion axiom* $w \dot{\sqsubseteq} R$*, written* $\mathcal{I} \models w \dot{\sqsubseteq} R$*, if* $w^\mathcal{I} \subseteq R^\mathcal{I}$*. An interpretation is a* ***model*** *of a role hierarchy* R_h *if it satisfies all RIAs in* R_h*, written* $\mathcal{I} \models R_h$*.*

A RIA $w \dot{\sqsubseteq} R$ *is* $\prec$***-regular*** *if* R *is a role name, and*

1. $w = RR$*, or*
2. $w = R^-$*, or*
3. $w = S_1 \ldots S_n$ *and* $S_i \prec R$*, for all* $1 \leq i \leq n$*, or*
4. $w = RS_1 \ldots S_n$ *and* $S_i \prec R$*, for all* $1 \leq i \leq n$*, or*
5. $w = S_1 \ldots S_n R$ *and* $S_i \prec R$*, for all* $1 \leq i \leq n$*.*

Finally, a role hierarchy $\mathcal{R}_h$ *is* ***regular*** *if there exists a regular order* $\prec$ *such that each RIA in* $\mathcal{R}_h$ *is* $\prec$*-regular.*

Regularity prevents a role hierarchy from containing cyclic dependencies. For instance, the role hierarchy

$$\{RS \dot{\sqsubseteq} S, \quad RT \dot{\sqsubseteq} R, \quad VT \dot{\sqsubseteq} T, \quad VS \dot{\sqsubseteq} V\}$$

is not regular because it would require $\prec$ to satisfy $S \prec V \prec T \prec R \prec S$, which would imply $S \prec S$, thus contradicting the irreflexivity of $\prec$. Such cyclic dependencies are known to lead to undecidability (Horrocks & Sattler, 2004).

Also, note that RIAs of the form $RR^- \dot{\sqsubseteq} R$, which would imply (a weak form of) reflexivity of R, are not regular according to the definition of regular orderings. However, the same condition on R can be imposed by using the GCI $\exists R.\top \dot{\sqsubseteq} \exists R.\mathsf{Self}$; see below.

From the definition of the semantics of inverse roles, it follows immediately that $\langle x, y\rangle \in w^\mathcal{I}$ iff $\langle y, x\rangle \in \mathsf{Inv}(w)^\mathcal{I}$. Hence, each model satisfying $w \dot{\sqsubseteq} S$ also satisfies $\mathsf{Inv}(w) \dot{\sqsubseteq} \mathsf{Inv}(S)$ (and vice versa), and thus the restriction to those

RIAs with only role *names* on their right hand side does not have any effect on expressivity.

Given a role hierarchy $\mathcal{R}_h$, we define the relation $\dot{\sqsubseteq}^*$ to be the transitive-reflexive closure of $\dot{\sqsubseteq}$ over $\{R \dot{\sqsubseteq} S, \mathsf{Inv}(R) \dot{\sqsubseteq} \mathsf{Inv}(S) \mid R \dot{\sqsubseteq} S \in \mathcal{R}_h\}$. A role R is called a **sub-role** (**super-role**) of a role S if $R \dot{\sqsubseteq}^* S$ ($S \dot{\sqsubseteq}^* R$). Two roles R and S are **equivalent** ($R \equiv S$) if $R \dot{\sqsubseteq}^* S$ and $S \dot{\sqsubseteq}^* R$.

Note that, due to restriction (3) in the definition of $\prec$-regularity, we also restrict $\dot{\sqsubseteq}^*$ to be acyclic, and thus regular role hierarchies never contain two equivalent roles.[2]

Next, let us turn to the second component of Rboxes, the role assertions. For an interpretation $\mathcal{I}$, we define $Diag^{\mathcal{I}}$ to be the set $\{\langle x, x\rangle \mid x \in \Delta^{\mathcal{I}}\}$. Note that, since the interpretation of the universal role U is fixed in any given model (as the universal relation on $\Delta^I \times \Delta^I$ which is, by definition, reflexive, symmetric, and transitive), we disallow the universal role to appear in role assertions.

Definition 3 (Role Assertions) *For roles $R, S \neq U$, we call the assertions $\mathsf{Ref}(R)$, $\mathsf{Irr}(R)$, $\mathsf{Sym}(R)$, $\mathsf{Asy}(R)$, $\mathsf{Tra}(R)$, and $\mathsf{Dis}(R, S)$, **role assertions**, where, for each interpretation $\mathcal{I}$ and all $x,\ y,\ z \in \Delta^{\mathcal{I}}$, we have:*

$\mathcal{I} \models \mathsf{Sym}(R)$	*if*	$\langle x, y\rangle \in R^{\mathcal{I}}$ *implies* $\langle y, x\rangle \in R^I$;
$\mathcal{I} \models \mathsf{Asy}(R)$	*if*	$\langle x, y\rangle \in R^{\mathcal{I}}$ *implies* $\langle y, x\rangle \notin R^{\mathcal{I}}$
$\mathcal{I} \models \mathsf{Tra}(R)$	*if*	$\langle x, y\rangle \in R^{\mathcal{I}}$ *and* $\langle y, z\rangle \in R^{\mathcal{I}}$ *imply* $\langle x, z\rangle \in R^I$;
$\mathcal{I} \models \mathsf{Ref}(R)$	*if*	$Diag^{\mathcal{I}} \subseteq R^{\mathcal{I}}$;
$\mathcal{I} \models \mathsf{Irr}(R)$	*if*	$R^{\mathcal{I}} \cap Diag^{\mathcal{I}} = \emptyset$;
$\mathcal{I} \models \mathsf{Dis}(R, S)$	*if*	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$.

Adding symmetric and transitive role assertions is a trivial move since both of these expressive means can be replaced with complex role inclusion axioms as follows: $\mathsf{Sym}(R)$ is equivalent to $R^- \dot{\sqsubseteq} R$ and $\mathsf{Tra}(R)$ is equivalent to $RR \dot{\sqsubseteq} R$.

Thus, as far as expressivity is concerned, we can assume, for convenience, that no role assertions of the form $\mathsf{Tra}(R)$ or $\mathsf{Sym}(R)$ appear in $\mathcal{R}_a$, but that transitive and/or symmetric roles will be handled by the RIAs alone. In particular, notice that regularity of a role hierarchy is preserved when replacing such role assertions with the corresponding RIAs.

The situation is different, however, for the other Rbox assertions. None of reflexivity, irreflexivity, antisymmetry or disjointness of roles can be enforced by role inclusion axioms. However, as we shall see later, reflexivity and irreflexivity of roles are closely related to the new concept $\exists R.\mathsf{Self}$.

Note that the version of antisymmetry introduced above is *strict* in the sense that it also implies irreflexivity as opposed to the more widely used notion of antisymmetry which allows for reflexive points. For instance, in *mereology*, the relation `PartOf` is usually assumed to be 'reflexive antisymmetric' (i.e., reflexivity, plus xRy and yRx implies $x = y$), while the relation `properPartOf` is assumed to be 'irreflexive antisymmetric' (defined just as antisymmetry above) (Simons, 1987; Casati & Varzi, 1999). The more general version of antisymmetry is more difficult to handle algorithmically, and we leave this to future work.

In $\mathcal{SHIQ}$ (and $\mathcal{SHOIQ}$), the application of qualified number restrictions has to be restricted to certain roles, called *simple roles*, to preserve decidability (Horrocks, Sattler, & Tobies, 1999). In the context of $\mathcal{SROIQ}$, the definition of *simple role* has to be slightly modified, and simple roles figure not only in qualified number restrictions, but in several other constructs as well. Intuitively, non-simple roles are those that are implied by the composition of roles.

Given a role hierarchy $\mathcal{R}_h$ and a set of role assertions $\mathcal{R}_a$ (without transitivity or symmetry assertions), the set of roles that are **simple in** $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$ is inductively defined as follows:

- a role name is simple if it does not occur on the right hand side of a RIA in $\mathcal{R}_h$,
- an inverse role R^- is simple if R is, and
- if R occurs on the right hand side of a RIA in $\mathcal{R}_h$, then R is simple if, for each $w \dot{\sqsubseteq} R \in \mathcal{R}_h$, $w = S$ for a simple role S.

A set of role assertions $\mathcal{R}_a$ is called **simple** if all roles R, S appearing in role assertions of the form $\mathsf{Irr}(R)$, $\mathsf{Asy}(R)$, or $\mathsf{Dis}(R, S)$, are simple in $\mathcal{R}$. If $\mathcal{R}$ is clear from the context, we often use "simple" instead of "simple in $\mathcal{R}$".

Definition 4 (Role Box) *A $\mathcal{SROIQ}$-**role box** (Rbox for short) is a set $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$, where $\mathcal{R}_h$ is a regular role hierarchy and $\mathcal{R}_a$ is a finite, simple set of role assertions.*

*An interpretation **satisfies a role box** $\mathcal{R}$ (written $\mathcal{I} \models \mathcal{R}$) if $\mathcal{I} \models R_h$ and $\mathcal{I} \models \phi$ for all role assertions $\phi \in R_a$. Such an interpretation is called a model of $\mathcal{R}$.*

Concepts and Inference Problems for $\mathcal{SROIQ}$

Definition 5 ($\mathcal{SROIQ}$ Concepts, Tboxes, and Aboxes) *The set of $\mathcal{SROIQ}$-**concepts** is the smallest set such that*

- *every concept name (including nominals) and $\top, \bot$ are concepts, and,*
- *if C, D are concepts, R is a role (possibly inverse), S is a simple role (possibly inverse), and n is a non-negative integer, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall R.C$, $\exists R.C$, $\exists S.\mathsf{Self}$, $(\geqslant nS.C)$, and $(\leqslant nS.C)$ are also concepts.*

*A **general concept inclusion axiom** (GCI) is an expression of the form $C \dot{\sqsubseteq} D$ for two $\mathcal{SROIQ}$-concepts C and D. A **Tbox** $\mathcal{T}$ is a finite set of GCIs.*

*An **individual assertion** is of one of the following forms: $a : C$, $(a, b) : R$, $(a, b) : \neg R$, or $a \neq b$, for $a, b \in \mathbf{I}$ (the set of individual names), a (possibly inverse) role R, and a $\mathcal{SROIQ}$-concept C. A $\mathcal{SROIQ}$-**Abox** $\mathcal{A}$ is a finite set of individual assertions.*

It is part of future work to determine which of the restrictions to simple roles in role assertions $\mathsf{Dis}(R, S)$, $\mathsf{Asy}(R)$, and $\mathsf{Irr}(R)$, as well as the concept expression $\exists S.\mathsf{Self}$, are strictly necessary in order to preserve decidability or practicability. These restrictions, however, allow a rather smooth integration of the new constructs into existing algorithms.

[2] This is not a serious restriction for, if $\mathcal{R}$ contains $\dot{\sqsubseteq}^*$ cycles, we can simply choose one role R from each cycle and replace all other roles in this cycle with R in the input Rbox, Tbox, and Abox.

Definition 6 (Semantics and Inference Problems) *Given an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, concepts C, D, roles R, S, and non-negative integers n, the **extension of complex concepts** is defined inductively by the following equations, where $\sharp M$ denotes the cardinality of a set M, and concept names, roles, and nominals are interpreted as in Definition 1:*

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}, \qquad \bot^{\mathcal{I}} = \emptyset, \qquad (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$$
$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$
$$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$$
$$(\exists R.\mathsf{Self})^{\mathcal{I}} = \{x \mid \langle x, x\rangle \in R^{\mathcal{I}}\}$$
$$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}$$
$$(\geqslant n R.C)^{\mathcal{I}} = \{x \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geqslant n\}$$
$$(\leqslant n R.C)^{\mathcal{I}} = \{x \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leqslant n\}$$

*$\mathcal{I}$ is a **model of a Tbox** $\mathcal{T}$ (written $\mathcal{I} \models \mathcal{T}$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for each GCI $C \dot{\sqsubseteq} D$ in $\mathcal{T}$. A concept C is called **satisfiable** if there is an interpretation $\mathcal{I}$ with $C^{\mathcal{I}} \neq \emptyset$. A concept D **subsumes** a concept C (written $C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each interpretation. For an interpretation $\mathcal{I}$, an element $x \in \Delta^{\mathcal{I}}$ is called an **instance** of a concept C if $x \in C^{\mathcal{I}}$.*

*$\mathcal{I}$ **satisfies** (is a model of) **an Abox** $\mathcal{A}$ ($\mathcal{I} \models \mathcal{A}$) if for all individual assertions $\phi \in \mathcal{A}$ we have $\mathcal{I} \models \phi$, where*

$$\mathcal{I} \models a\!:\!C \quad \text{if} \quad a^{\mathcal{I}} \in C^{\mathcal{I}};$$
$$\mathcal{I} \models a \not\doteq b \quad \text{if} \quad a^{\mathcal{I}} \neq b^{\mathcal{I}};$$
$$\mathcal{I} \models (a, b)\!:\!R \quad \text{if} \quad \langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}};$$
$$\mathcal{I} \models (a, b)\!:\!\neg R \quad \text{if} \quad \langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \notin R^{\mathcal{I}}.$$

*An Abox $\mathcal{A}$ is **consistent** with respect to an Rbox $\mathcal{R}$ and a Tbox $\mathcal{T}$ if there is a model $\mathcal{I}$ for $\mathcal{R}$ and $\mathcal{T}$ such that $\mathcal{I} \models \mathcal{A}$.*

The above inference problems can be defined w.r.t. a role box $\mathcal{R}$ and/or a Tbox $\mathcal{T}$ in the usual way, i.e., by replacing *interpretation* with *model of* $\mathcal{R}$ and/or $\mathcal{T}$.

Reduction of Inference Problems

For DLs that are closed under negation, subsumption and (un)satisfiability of concepts can be mutually reduced: $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable, and C is unsatisfiable iff $C \sqsubseteq \bot$. Furthermore, a concept C is satisfiable iff the Abox $\{a\!:\!C\}$ (a a 'new' individual name) is consistent.

It is straightforward to extend these reductions to Rboxes and Tboxes. In contrast, the reduction of inference problems w.r.t. a Tbox to pure concept inference problems (possibly w.r.t. a role hierarchy), deserves special care: in Baader (1991); Schild (1991); Baader *et al.* (1993), the *internalisation* of GCIs is introduced, a technique that realises exactly this reduction. For $\mathcal{SROIQ}$, this technique only needs to be slightly modified. We will show in a series of steps that, in $\mathcal{SROIQ}$, satisfiability of a concept C with respect to a triple $\langle \mathcal{A}, \mathcal{R}, \mathcal{T}\rangle$ of, respectively, a $\mathcal{SROIQ}$ Abox, Rbox, and Tbox, can be reduced to concept satisfiability of a concept C' with respect to an Rbox $\mathcal{R}'$, where the Rbox $\mathcal{R}'$ only contains role assertions of the form $\mathsf{Dis}(R, S)$, $\mathsf{Ref}(R)$, or $\mathsf{Asy}(R)$, and the universal role U does not appear in C'.

While nominals can be used to 'internalise' the Abox, in order to eliminate the universal role, we use a 'simulated' universal role U', i.e., a reflexive, symmetric, and transitive super-role of all roles and their inverses appearing in $\langle \mathcal{A}, \mathcal{R}, \mathcal{T}\rangle$, and which, additionally, connects all nominals appearing in the input.

Thus, let C and $\langle \mathcal{A}, \mathcal{R}, \mathcal{T}\rangle$ be a $\mathcal{SROIQ}$ concept and Abox, Rbox, and Tbox, respectively. In a first step, we replace the Abox $\mathcal{A}$ with an Abox $\mathcal{A}'$ such that $\mathcal{A}'$ only contains individual assertions of the form $a\!:\!C$. To this purpose, we associate with every individual $a \in \mathbf{I}$ appearing in $\mathcal{A}$ a new nominal o_a not appearing in $\mathcal{T}$ or C. Next, $\mathcal{A}'$ is the result of replacing every individual assertion in $\mathcal{A}$ of the form $(a, b) : R$ with $a : \exists R.o_b$, every $(a, b) : \neg R$ with $a : \forall R.\neg o_b$, and every $a \not\doteq b$ with $a : \neg o_b$. Now, given C and $\mathcal{A}'$, define C' as follows:

$$C' := C \sqcap \bigsqcap_{a:D \in \mathcal{A}'} \exists U.(o_a \sqcap D),$$

where U is the universal role. It should be clear that C is satisfiable with respect to $\langle \mathcal{A}, \mathcal{R}, \mathcal{T}\rangle$ if and only if C' is satisfiable with respect to $\langle \mathcal{R}, \mathcal{T}\rangle$.

Lemma 7 (Abox Elimination) *$\mathcal{SROIQ}$ concept satisfiability with respect to Aboxes, Rboxes, and Tboxes is polynomially reducible to $\mathcal{SROIQ}$ concept satisfiability with respect to Rboxes and Tboxes only.*

Hence, in the following we will assume that Aboxes have been eliminated. Next, although we have the 'real' universal role U present in the language, the following lemma shows how general concept inclusion axioms can be *internalised* while at the same time eliminating occurrences of the universal role U, using a simulated "universal" role U', that is, a transitive super-role of all roles (except U) occurring in $\mathcal{T}$ or $\mathcal{R}$ and their respective inverses. Recall that the universal role U is not allowed to appear in Rboxes.

Lemma 8 (Tbox and Universal Role Elimination) *Let C, D be concepts, $\mathcal{T}$ a Tbox, and $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$ an Rbox. Let $U' \neq U$ be a role that does not occur in C, D, $\mathcal{T}$, or $\mathcal{R}$, and, for X a Tbox or a concept, let X' result from X by replacing every occurrence of U with U'. We define*

$$C_{\mathcal{T}'} := \forall U'.\Big(\bigsqcap_{C_i' \dot{\sqsubseteq} D_i' \in \mathcal{T}'} \neg C_i' \sqcup D_i'\Big) \sqcap \Big(\bigsqcap_{\mathbf{N} \ni o \in \mathcal{T} \cup C \cup D} \exists U'.o\Big),$$

$$\mathcal{R}_h^{U'} := \mathcal{R}_h \cup \{R \dot{\sqsubseteq} U' \mid R \text{ occurs in } C', D', \mathcal{T}', \text{ or } \mathcal{R}\},$$

$$\mathcal{R}_a^{U'} := \mathcal{R}_a \cup \{\mathsf{Tra}(U'), \mathsf{Sym}(U'), \mathsf{Ref}(U')\}, \text{ and}$$

$$\mathcal{R}_{U'} := \mathcal{R}_h^{U'} \cup \mathcal{R}_a^{U'}. \text{ Then}$$

- *C is satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{R}$ iff $C' \sqcap C_{\mathcal{T}'}$ is satisfiable w.r.t. $\mathcal{R}_{U'}$.*
- *D subsumes C with respect to $\mathcal{T}$ and $\mathcal{R}$ iff $C' \sqcap \neg D' \sqcap C_{\mathcal{T}'}$ is unsatisfiable w.r.t. $\mathcal{R}_{U'}$.*

The proof of Lemma 8 is similar to the ones that can be found in Schild (1991) and Baader (1991). Most importantly, it must be shown that (a): if a $\mathcal{SROIQ}$-concept C is satisfiable with respect to a Tbox $\mathcal{T}$ and an Rbox $\mathcal{R}$, then $C, \mathcal{T}, \mathcal{R}$ have a **nominal connected** model, i.e., a model which is a union of connected components, where each such component contains a nominal, and where any two elements

of a connected component are connected by a role path over those roles occurring in C, $\mathcal{T}$ or $\mathcal{R}$, and (b): if y is reachable from x via a role path (possibly involving inverse roles), then $\langle x, y\rangle \in U'^{\mathcal{I}}$. These are easy consequences of the semantics and the definition of U' and $C_{\mathcal{T}'}$, which guarantees that all nominals are connected by U' links.

Now, note also that, instead of having a role assertion $\mathsf{Irr}(R) \in \mathcal{R}_a$, we can add, equivalently, the GCI $\top \dot{\sqsubseteq} \neg\exists R.\mathsf{Self}$ to $\mathcal{T}$, which can in turn be internalised. Likewise, instead of asserting $\mathsf{Ref}(R)$, we can, equivalently, add the GCI $\top \dot{\sqsubseteq} \exists R.\mathsf{Self}$ to $\mathcal{T}$. However, in the case of $\mathsf{Ref}(R)$ this replacement is only admissible for *simple* roles R and thus not possible (syntactically) in general.

Thus, using these equivalences (including the replacement of Rbox assertions of the form $\mathsf{Sym}(R)$ and $\mathsf{Tra}(R)$) and Lemmas 7 and 8, we arrive at the following theorem:

Theorem 9 (Reduction)

1. *Satisfiability and subsumption of $\mathcal{SROIQ}$-concepts w.r.t. Tboxes, Aboxes, and Rboxes, are polynomially reducible to (un)satisfiability of $\mathcal{SROIQ}$-concepts w.r.t. Rboxes.*
2. *W.l.o.g., we can assume that Rboxes do not contain role assertions of the form $\mathsf{Irr}(R)$, $\mathsf{Tra}(R)$, or $\mathsf{Sym}(R)$, and that the universal role is not used.*

With Theorem 9, all standard inference problems for $\mathcal{SROIQ}$-concepts and Aboxes can be reduced to the problem of determining the consistency of a $\mathcal{SROIQ}$-concept w.r.t. to an Rbox (both not containing the universal role), where we can assume w.l.o.g. that all role assertions in the Rbox are of the form $\mathsf{Ref}(R)$, $\mathsf{Asy}(R)$, or $\mathsf{Dis}(R, S)$—we call such an Rbox **reduced**.

$\mathcal{SROIQ}$ is Decidable

In this section, we show that $\mathcal{SROIQ}$ is decidable. We present a tableau-based algorithm that decides the consistency of a $\mathcal{SROIQ}$ concept w.r.t. a reduced Rbox, and therefore also all standard inference problems as discussed above, see Theorem 9. Therefore, in the following, by Rbox we always mean *reduced* Rbox.

The algorithm tries to construct, for a $\mathcal{SROIQ}$-concept C and an Rbox $\mathcal{R}$, a *tableau* for C and $\mathcal{R}$, that is, an abstraction of a model of C and $\mathcal{R}$.

For a regular role hierarchy $\mathcal{R}_h$ and a (possibly inverse) role S occurring in $\mathcal{R}_h$, a non-deterministic finite automaton (NFA) $\mathcal{B}_S$ is defined. The construction of these automata is identical to the one presented in Horrocks & Sattler (2004), and we therefore refer to this paper for detailed definitions and proofs of the automata related results below.

The following proposition states that $\mathcal{B}_S$ indeed captures all implications between (paths of) roles and S that are consequences of the role hierarchy $\mathcal{R}_h$, where $L(\mathcal{B}_S)$ denotes the language (a set of strings of roles) accepted by $\mathcal{B}_S$.

Proposition 10 *$\mathcal{I}$ is a model of $\mathcal{R}_h$ if and only if, for each (possibly inverse) role S occurring in $\mathcal{R}_h$, each word $w \in L(\mathcal{B}_S)$, and each $\langle x, y\rangle \in w^{\mathcal{I}}$, we have $\langle x, y\rangle \in S^{\mathcal{I}}$.*

Unfortunately, as shown in Horrocks & Sattler (2004), the size of $\mathcal{B}_R$ can be exponential in the size of $\mathcal{R}$. Horrocks & Sattler (2004) consider certain further syntactic restrictions of role hierarchies (there called *simple* role hierarchies) that avoid this exponential blow-up. We conjecture that, without some such further restriction, this blow-up is unavoidable. The following technical Lemma from Horrocks & Sattler (2004) will be needed later on.

Lemma 11

1. *$S \in L(\mathcal{B}_S)$ and, if $w \dot{\sqsubseteq} S \in \mathcal{R}$, then $w \in L(\mathcal{B}_S)$.*
2. *If S is a simple role, then $L(\mathcal{B}_S) = \{R \mid R \sqsubseteq S\}$.*
3. *$L(\mathcal{B}_{\mathsf{Inv}(S)}) = \{\mathsf{Inv}(w) \mid w \in L(\mathcal{B}_S)\}$.*

A Tableau for $\mathcal{SROIQ}$

In the following, if not stated otherwise, C, D (possibly with subscripts) denote $\mathcal{SROIQ}$-concepts (not using the universal role), R, S (possibly with subscripts) roles, $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$ an Rbox, and $\mathbf{R}_C$ the set of roles occurring in C and $\mathcal{R}$ together with their inverses. Furthermore, as noted in Theorem 9, we can (and will from now on) assume w.l.o.g. that all role assertions appearing in $\mathcal{R}_a$ are of the form $\mathsf{Dis}(R, S)$, $\mathsf{Asy}(R)$, or $\mathsf{Ref}(R)$.

We start by defining $\mathsf{fclos}(C_0, \mathcal{R})$, the *closure* of a concept C_0 w.r.t. a regular role hierarchy $\mathcal{R}$. Intuitively, this contains all relevant sub-concepts of C_0 together with universal value restrictions over sets of role paths described by an NFA. We use NFAs in universal value restrictions to memorise the path between an object that has to satisfy a value restriction and other objects. To do this, we "push" this NFA-value restriction along all paths while the NFA gets "updated" with the path taken so far. For this "update", we use the following definition.

Definition 12 *For $\mathcal{B}$ an NFA and q a state of $\mathcal{B}$, $\mathcal{B}(q)$ denotes the NFA obtained from $\mathcal{B}$ by making q the (only) initial state of $\mathcal{B}$, and we use $q \xrightarrow{S} q' \in \mathcal{B}$ to denote that $\mathcal{B}$ has a transition $q \xrightarrow{S} q'$.*

Without loss of generality, we assume all concepts to be in **negation normal form** (NNF), that is, negation occurs only in front of concept names or in front of $\exists R.\mathsf{Self}$. Any $\mathcal{SROIQ}$-concept can easily be transformed into an equivalent one in NNF by pushing negations inwards using a combination of De Morgan's laws and equivalences such as $\neg(\exists R.C) \equiv (\forall R.\neg C)$, $\neg(\leqslant nR.C) \equiv (\geqslant (n+1)R.C)$, etc. We use $\dot{\neg}C$ for the NNF of $\neg C$. Obviously, the length of $\dot{\neg}C$ is linear in the length of C.

For a concept C_0, $\mathsf{clos}(C_0)$ is the smallest set that contains C_0 and that is closed under sub-concepts and $\dot{\neg}$. The set $\mathsf{fclos}(C_0, \mathcal{R})$ is then defined as follows:

$$\mathsf{fclos}(C_0, \mathcal{R}) := \mathsf{clos}(C_0) \cup \{\forall \mathcal{B}_S(q).D \mid \forall S.D \in \mathsf{clos}(C_0) \text{ and } \mathcal{B}_S \text{ has a state } q\}.$$

It is not hard to show and well-known that the size of $\mathsf{clos}(C_0)$ is linear in the size of C_0. For the size of $\mathsf{fclos}(C_0, \mathcal{R})$, we have mentioned above that, for a role S,

the size of $\mathcal{B}_S$ can be exponential in the depth of $\mathcal{R}$. Since there are at most linearly many concepts $\forall S.D$, this yields a bound for the cardinality of $\mathsf{fclos}(C_0, \mathcal{R})$ that is exponential in the depth of $\mathcal{R}$ and linear in the size of C_0.

Definition 13 (Tableau) *$T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ is a* tableau *for C_0 w.r.t. $\mathcal{R}$ if*

- $\mathbf{S}$ *is a non-empty set;*
- $\mathcal{L} : \mathbf{S} \to 2^{\mathsf{fclos}(C_0,\mathcal{R})}$ *maps each element in $\mathbf{S}$ to a set of concepts;*
- $\mathcal{E} : \mathbf{R}_{C_0} \to 2^{\mathbf{S}\times\mathbf{S}}$ *maps each role to a set of pairs of elements in $\mathbf{S}$;*
- $C_0 \in \mathcal{L}(s)$ *for some $s \in \mathbf{S}$.*

Furthermore, for all $s, t \in \mathbf{S}$, $C, C_1, C_2 \in \mathsf{fclos}(C_0, \mathcal{R})$, $o \in \mathbf{N} \cap \mathsf{fclos}(C_0, \mathcal{R})$, $R, S \in \mathbf{R}_{C_0}$, and

$$S^T(s, C) := \{r \in \mathbf{S} \mid \langle s, r\rangle \in \mathcal{E}(S) \text{ and } C \in \mathcal{L}(r)\},$$

the tableau T satisfies:

(P1) $C \in \mathcal{L}(s) \implies \neg C \notin \mathcal{L}(s)$, *($C$ atomic or $\exists R.\mathsf{Self}$);*
(P2) $\top \in \mathcal{L}(s)$, *and* $\bot \notin \mathcal{L}(s)$;
(P3) $\exists R.\mathsf{Self} \in \mathcal{L}(s) \implies \langle s, s\rangle \in \mathcal{E}(R)$;
(P4) $\neg\exists R.\mathsf{Self} \in \mathcal{L}(s) \implies \langle s, s\rangle \notin \mathcal{E}(R)$;
(P5) $C_1 \sqcap C_2 \in \mathcal{L}(s) \implies C_1, C_2 \in \mathcal{L}(s)$;
(P6) $C_1 \sqcup C_2 \in \mathcal{L}(s) \implies C_1 \in \mathcal{L}(s)$ *or* $C_2 \in \mathcal{L}(s)$;
(P7) $\forall\mathcal{B}(p).C \in \mathcal{L}(s)$, $\langle s, t\rangle \in \mathcal{E}(S)$, *and* $p \xrightarrow{S} q \in \mathcal{B}(p) \implies \forall\mathcal{B}(q).C \in \mathcal{L}(t)$;
(P8) $\forall\mathcal{B}.C \in \mathcal{L}(s)$ *and* $\varepsilon \in L(\mathcal{B}) \implies C \in \mathcal{L}(s)$;
(P9) $\forall S.C \in \mathcal{L}(s) \implies \forall\mathcal{B}_S.C \in \mathcal{L}(s)$;
(P10) $\exists S.C \in \mathcal{L}(s) \implies$ *there is some $r \in \mathbf{S}$ with* $\langle s, r\rangle \in \mathcal{E}(S)$ *and* $C \in \mathcal{L}(r)$;
(P11) $\langle s, t\rangle \in \mathcal{E}(R) \iff \langle t, s\rangle \in \mathcal{E}(\mathsf{Inv}(R))$;
(P12) $\langle s, t\rangle \in \mathcal{E}(R)$ *and* $R \sqsubseteq^* S \implies \langle s, t\rangle \in \mathcal{E}(S)$;
(P13) $(\leqslant n S.C) \in \mathcal{L}(s) \implies \sharp S^T(s, C) \leqslant n$;
(P14) $(\geqslant n S.C) \in \mathcal{L}(s) \implies \sharp S^T(s, C) \geqslant n$;
(P15) $(\leqslant n S.C) \in \mathcal{L}(s)$ *and* $\langle s, t\rangle \in \mathcal{E}(S) \implies C \in \mathcal{L}(t)$ *or* $\dot\neg C \in \mathcal{L}(t)$;
(P16) $\mathsf{Dis}(R, S) \in \mathcal{R}_a \implies \mathcal{E}(R) \cap \mathcal{E}(S) = \emptyset$;
(P17) $\mathsf{Ref}(R) \in \mathcal{R}_a \implies \langle s, s\rangle \in \mathcal{E}(R)$;
(P18) $\mathsf{Asy}(R) \in \mathcal{R}_a \implies \langle s, t\rangle \in \mathcal{E}(R)$ *implies* $\langle t, s\rangle \notin \mathcal{E}(R)$;
(P19) $o \in \mathcal{L}(r)$ *for some $r \in \mathbf{S}$;*
(P20) $o \in \mathcal{L}(s) \cap \mathcal{L}(t) \implies s = t$.

Theorem 14 (Tableau) *A $\mathcal{SROIQ}$-concept C_0 is satisfiable w.r.t. a reduced Rbox $\mathcal{R}$ iff there exists a tableau for C_0 w.r.t. $\mathcal{R}$.*

Proof: For the *if* direction, let $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ be a tableau for C_0 w.r.t. $\mathcal{R}$. We extend the relational structure of T and then prove that this indeed gives a model.

More precisely, a model $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ of C_0 and $\mathcal{R}$ can be defined as follows: we set $\Delta^\mathcal{I} := \mathbf{S}$, $C^\mathcal{I} := \{s \mid C \in \mathcal{L}(s)\}$ for concept names C in $\mathsf{fclos}(C_0, \mathcal{R})$, where (P19) and (P20) guarantee that nominals are indeed interpreted as singleton sets, and, for roles names $R \in \mathbf{R}_{C_0}$, we set

$$R^\mathcal{I} := \{\langle s_0, s_n\rangle \in (\Delta^\mathcal{I})^2 \mid \text{exists } s_1, \ldots, s_{n-1} \text{ with } \langle s_i, s_{i+1}\rangle \in \mathcal{E}(S_{i+1}) \text{ and } S_1 \cdots S_n \in L(\mathcal{B}_R)\}$$

The semantics of complex concepts is given through the definition of the $\mathcal{SROIQ}$-semantics. Due to Lemma 11.3 and (P11), the semantics of inverse roles can either be given directly as for role names, or by setting $(R^-)^\mathcal{I} := \{\langle y, x\rangle \mid \langle x, y\rangle \in R^\mathcal{I}\}$. Moreover, we have, by definition of $\mathcal{I}$, Lemma 11.2, (P11), and (P12) that, for T a simple role, $T^\mathcal{I} = \mathcal{E}(T)$.

We have to show that $\mathcal{I}$ is a model of $\mathcal{R}$ and C_0. We begin by showing that $\mathcal{I} \models \mathcal{R}$. Since $\mathcal{R}$ is reduced, we only have to deal with role assertions of the form $\mathsf{Dis}(R, S)$, $\mathsf{Ref}(R)$, and $\mathsf{Asy}(R)$.

Consider an assertion $\mathsf{Dis}(R, S) \in \mathcal{R}$. By definition of $\mathcal{SROIQ}$-Rboxes, both R and S are simple roles, and thus $R^\mathcal{I} = \mathcal{E}(R)$ and $S^\mathcal{I} = \mathcal{E}(S)$. Moreover, (P16) implies $\mathcal{E}(R) \cap \mathcal{E}(S) = \emptyset$, and thus $R^\mathcal{I} \cap S^\mathcal{I} = \emptyset$. Next, if $\mathsf{Ref}(R) \in \mathcal{R}_a$, (P17) and $R \in L(\mathcal{B}_R)$ (Lemma 11.1) imply $Diag^\mathcal{I} \subseteq R^\mathcal{I}$. Finally, if $\mathsf{Asy}(R) \in \mathcal{R}_a$ then $R^\mathcal{I} = \mathcal{E}(R)$ since R is simple, and so $\langle s, t\rangle \in R^\mathcal{I}$ implies $\langle s, t\rangle \in \mathcal{E}(R)$ and so $\langle t, s\rangle \notin \mathcal{E}(R)$ by (P18), whence $\langle t, s\rangle \notin R^\mathcal{I}$. Thus $\mathcal{I}$ satisfies each role assertion in $\mathcal{R}_a$.

Next, we have to show that $\mathcal{I} \models \mathcal{R}_h$. Due to Proposition 10, it suffices to prove that, for each (possibly inverse) role S, each word $w \in L(\mathcal{B}_S)$, and each $\langle x, y\rangle \in w^\mathcal{I}$, we have $\langle x, y\rangle \in S^\mathcal{I}$. The proof of this is identical to the case of $\mathcal{RIQ}$ and can be found in Horrocks & Sattler (2004).

Secondly, we prove that $\mathcal{I}$ is a model of C_0. We show that $C \in \mathcal{L}(s)$ implies $s \in C^\mathcal{I}$ for each $s \in \mathbf{S}$ and each $C \in \mathsf{fclos}(\mathcal{A}, \mathcal{R})$. This proof can be given by induction on the length of concepts, where we count neither negation nor integers in number restrictions. The only interesting cases are $C = (\leqslant n S.E)$, $C = \forall S.E$, and $C = (\neg)\exists R.\mathsf{Self}$ (for the other cases, see Horrocks, Sattler, & Tobies (2000); Horrocks & Sattler (2002)):

- If $(\leqslant n S.E) \in \mathcal{L}(s)$, then (P13) implies that $\#S^T(s, E) \leq n$. Moreover, since S is simple, Lemma 11.2 implies that $L(\mathcal{B}_S) = \{S' \mid S' \sqsubseteq^* S\}$, and (P12) implies that $S^\mathcal{I} = \mathcal{E}(S)$. Hence (P15) implies that, for all t, if $\langle s, t\rangle \in S^\mathcal{I}$, then $E \in \mathcal{L}(t)$ or $\dot\neg E \in \mathcal{L}(t)$. By induction $E^\mathcal{I} = \{t \mid E \in \mathcal{L}(t)\}$, and thus $s \in (\leqslant n S.E)^\mathcal{I}$.
- Let $\forall S.E \in \mathcal{L}(s)$ and $\langle s, t\rangle \in S^\mathcal{I}$. From (P9) we have that $\forall\mathcal{B}_S.E \in \mathcal{L}(s)$. By definition of $S^\mathcal{I}$, there are $S_1 \ldots S_n \in L(\mathcal{B}_S)$ and s_i with $s = s_0$, $t = s_n$, and $\langle s_{i-1}, s_i\rangle \in \mathcal{E}(S_i)$. Applying (P7) n times, this yields $\forall\mathcal{B}_S(q).E \in \mathcal{L}(t)$ for q a final state of $\mathcal{B}_S$. Thus (P8) implies that $E \in \mathcal{L}(t)$. By induction, $t \in E^\mathcal{I}$, and thus $s \in (\forall S.E)^\mathcal{I}$.
- Let $\exists R.\mathsf{Self} \in \mathcal{L}(s)$. Then, by (P3), $\langle s, s\rangle \in \mathcal{E}(R)$ and, since $R \in L(B_R)$ and by definition of $\mathcal{I}$, we have $\langle s, s\rangle \in R^\mathcal{I}$. It follows that $s \in (\exists R.\mathsf{Self})^\mathcal{I}$.

- Let $\neg\exists R.\mathsf{Self} \in \mathcal{L}(s)$. Then, by (P4), $\langle s,s\rangle \notin \mathcal{E}(R)$. Since R is a simple role, $R^{\mathcal{I}} = \mathcal{E}(R)$. Hence $\langle s,s\rangle \notin R^{\mathcal{I}}$, and so $s \in (\neg\exists R.\mathsf{Self})^{\mathcal{I}}$.

For the converse, suppose $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model of C_0 w.r.t. $\mathcal{R}$. We define a tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ for C_0 and $\mathcal{R}$ as follows:

$$\begin{aligned}
\mathbf{S} &:= \Delta^{\mathcal{I}};\\
\mathcal{E}(R) &:= R^{\mathcal{I}};\ \text{and}\\
\mathcal{L}(s) &:= \{C \in \mathsf{fclos}(C_0,\mathcal{R}) \mid s \in C^{\mathcal{I}}\}\\
&\quad \cup \{\forall \mathcal{B}_S.C \mid \forall S.C \in \mathsf{fclos}(C_0,\mathcal{R}) \text{ and } s \in (\forall S.C)^{\mathcal{I}}\}\\
&\quad \cup \{\forall \mathcal{B}_R(q).C \in \mathsf{fclos}(C_0,\mathcal{R}) \mid S_1 \cdots S_n \in L(\mathcal{B}_R(q)) \Rightarrow\\
&\qquad s \in (\forall S_1.\forall S_2.\cdots\forall S_n.C)^{\mathcal{I}}, \varepsilon \in L(\mathcal{B}_R(q)) \Rightarrow s \in C^{\mathcal{I}}\}
\end{aligned}$$

We have to show that T satisfies (P1)–(P20), and restrict our attention to the only new cases.

For (P9), if $\forall S.C \in \mathcal{L}(s)$, then $s \in (\forall S.C)^{\mathcal{I}}$ and thus $\forall \mathcal{B}_S.C \in \mathcal{L}(s)$ by definition of T.

For (P7), let $\forall \mathcal{B}(p).C \in \mathcal{L}(s)$ and $\langle s,t\rangle \in \mathcal{E}(S)$. Assume that there is a transition $p \xrightarrow{S} q$ in $\mathcal{B}(p)$ and $\forall \mathcal{B}(q).C \notin \mathcal{L}(t)$. By definition of T, this can have two reasons:

1. there is a word $S_2 \dots S_n \in L(\mathcal{B}(q))$ and $t \notin (\forall S_2.\dots\forall S_n.C)^{\mathcal{I}}$. However, this implies that $SS_2 \dots S_n \in L(\mathcal{B}(p))$ and thus that we have $s \in (\forall S.\forall S_2.\dots\forall S_n.C)^{\mathcal{I}}$, which contradicts, together with $\langle s,t\rangle \in S^{\mathcal{I}}$, the definition of the semantics of $\mathcal{SROIQ}$ concepts.
2. $\varepsilon \in L(\mathcal{B}(q))$ and $t \notin C^{\mathcal{I}}$. This implies that $S \in L(\mathcal{B}(p))$ and thus contradicts $s \in (\forall S.C)^{\mathcal{I}}$.

For (P8), $\varepsilon \in L(\mathcal{B}(p))$ implies $s \in C^{\mathcal{I}}$ by definition of T, and thus $C \in \mathcal{L}(s)$.

Finally, (P16)–(P20) follow immediately from the definition of the semantics.

□

The Tableau Algorithm

In this section, we present a terminating, sound, and complete tableau algorithm that decides consistency of $\mathcal{SROIQ}$-concepts not using the universal role w.r.t. reduced Rboxes, and thus, using Theorem 9, also concept satisfiability w.r.t. Rboxes, Tboxes and Aboxes.

We first define the underlying data structures and corresponding operations. For more detailed explanations concerning the intuitions underlying these definitions, consult Horrocks & Sattler (2005).

The algorithm generates a *completion graph*, a structure that, if complete and clash-free, can be unravelled to a (possibly infinite) tableau for the input concept and Rbox. Moreover, it is shown that the algorithm returns a complete and clash-free completion graph for C_0 and $\mathcal{R}$ if and only if there exists a tableau for C_0 and $\mathcal{R}$, and thus with Lemma 14, if and only if the concept C_0 is satisfiable w.r.t. $\mathcal{R}$.

As usual, in the presence of transitive roles, *blocking* is employed to ensure termination of the algorithm (Horrocks, Sattler, & Tobies, 2000).

Definition 15 (Completion Graph) *Let $\mathcal{R}$ be a reduced Rbox, let C_0 be a $\mathcal{SROIQ}$-concept in NNF not using the universal role, and let $\mathbf{N}$ be the set of nominals. A **completion graph** for C_0 with respect to $\mathcal{R}$ is a directed graph $\mathbf{G} = (V, E, \mathcal{L}, \neq)$ where each node $x \in V$ is labelled with a set*

$$\begin{aligned}\mathcal{L}(x) \subseteq \mathsf{fclos}(C_0,\mathcal{R}) \cup \mathbf{N} \cup \{(\leqslant m R.C) \mid\\ (\leqslant n R.C) \in \mathsf{fclos}(C_0,\mathcal{R}) \text{ and } m \leq n\}\end{aligned}$$

and each edge $\langle x,y\rangle \in E$ is labelled with a set of role names $\mathcal{L}(\langle x,y\rangle)$ containing (possibly inverse) roles occurring in C_0 or $\mathcal{R}$. Additionally, we keep track of inequalities between nodes of the graph with a symmetric binary relation $\neq$ between the nodes of $\mathbf{G}$.

In the following, we often use $R \in \mathcal{L}(\langle x,y\rangle)$ as an abbreviation for $\langle x,y\rangle \in E$ and $R \in \mathcal{L}(\langle x,y\rangle)$.

*If $\langle x,y\rangle \in E$, then y is called a **successor** of x and x is called a **predecessor** of y. **Ancestor** is the transitive closure of predecessor, and **descendant** is the transitive closure of successor. A node y is called an R**-successor** of a node x if, for some R' with $R' \sqsubseteq^* R$, $R' \in \mathcal{L}(\langle x,y\rangle)$. A node y is called a **neighbour** (R-**neighbour**) of a node x if y is a successor (R-successor) of x or if x is a successor ($\mathsf{Inv}(R)$-successor) of y.*

For a role S and a node x in $\mathbf{G}$, we define the set of x's S-neighbours with C in their label, $S^{\mathbf{G}}(x,C)$, as follows:

$$S^{\mathbf{G}}(x,C) := \{y \mid y \text{ is an } S\text{-neighbour of } x \text{ and } C \in \mathcal{L}(y)\}.$$

*$\mathbf{G}$ is said to **contain a clash** if there are nodes x and y such that*

1. *$\bot \in \mathcal{L}(x)$, or*
2. *for some concept name A, $\{A, \neg A\} \subseteq \mathcal{L}(x)$, or*
3. *x is an S-neighbour of x and $\neg\exists S.\mathsf{Self} \in \mathcal{L}(x)$, or*
4. *there is some $\mathsf{Dis}(R,S) \in \mathcal{R}_a$ and y is an R- and an S-neighbour of x, or*
5. *there is some $\mathsf{Asy}(R) \in \mathcal{R}_a$ and y is an R-neighbour of x and x is an R-neighbour of y, or*
6. *there is some concept $(\leqslant n S.C) \in \mathcal{L}(x)$ and $\{y_0,\dots,y_n\} \subseteq S^{\mathbf{G}}(x,C)$ with $y_i \neq y_j$ for all $0 \leq i < j \leq n$, or*
7. *for some $o \in \mathbf{N}$, $x \neq y$ and $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$.*

If $o_1,\dots,o_\ell$ are all the nominals occurring in C_0 then the tableau algorithm is initialised with the completion graph $\mathbf{G} = (\{r_0, r_1 \dots, r_\ell\}, \emptyset, \mathcal{L}, \emptyset)$ with $\mathcal{L}(r_0) = \{C_0\}$ and $\mathcal{L}(r_i) = \{o_i\}$ for $1 \leq i \leq \ell$. $\mathbf{G}$ is then expanded by repeatedly applying the expansion rules given in Figure 1, stopping if a clash occurs.

Before describing the tableau algorithm in more detail, we define some terms and operations used in the (application of the) expansion rules:

Nominal Nodes and Blockable Nodes A node x is a **nominal node** if $\mathcal{L}(x)$ contains a nominal. A node that is not a nominal node is a **blockable** node. A nominal $o \in \mathbf{N}$ is said to be **new in $\mathbf{G}$** if no node in $\mathbf{G}$ has o in its label.

Blocking A node x is **label blocked** if it has ancestors x', y and y' such that

1. x is a successor of x' and y is a successor of y',
2. y, x and all nodes on the path from y to x are blockable,
3. $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x') = \mathcal{L}(y')$, *and*
4. $\mathcal{L}(\langle x', x\rangle) = \mathcal{L}(\langle y', y\rangle)$.

In this case, we say that y **blocks** x. A node is **blocked** if either it is label blocked or it is blockable and its predecessor is blocked; if the predecessor of a blockable node x is blocked, then we say that x is **indirectly blocked**.

Generating and Shrinking Rules and Safe Neighbours The $\geqslant$-, $\exists$- and *NN*-rules are called **generating rules**, and the $\leqslant$- and the o-rule are called **shrinking rules**. An R-neighbour y of a node x is **safe** if (i) x is blockable or if (ii) x is a nominal node and y is not blocked.

Pruning When a node y is **merged** into a node x, we "prune" the completion graph by removing y and, recursively, all blockable successors of y. More precisely, pruning a node y (written $\mathsf{Prune}(y)$) in $\mathbf{G} = (V, E, \mathcal{L}, \not\approx)$ yields a graph that is obtained from $\mathbf{G}$ as follows:

1. for all successors z of y, remove $\langle y, z\rangle$ from E and, if z is blockable, $\mathsf{Prune}(z)$;
2. remove y from V.

Merging In some rules, we "merge" one node into another node. Intuitively, when we merge a node y into a node x, we add $\mathcal{L}(y)$ to $\mathcal{L}(x)$, "move" all the edges leading *to* y so that they lead to x and "move" all the edges leading from y to nominal nodes so that they lead from x to the same nominal nodes; we then remove y (and blockable sub-trees below y) from the completion graph. More precisely, merging a node y into a node x (written $\mathsf{Merge}(y, x)$) in $\mathbf{G} = (V, E, \mathcal{L}, \not\approx)$ yields a graph that is obtained from $\mathbf{G}$ as follows:

1. for all nodes z such that $\langle z, y\rangle \in E$
   (a) if $\{\langle x, z\rangle, \langle z, x\rangle\} \cap E = \emptyset$, then add $\langle z, x\rangle$ to E and set $\mathcal{L}(\langle z, x\rangle) = \mathcal{L}(\langle z, y\rangle)$,
   (b) if $\langle z, x\rangle \in E$, then set $\mathcal{L}(\langle z, x\rangle) = \mathcal{L}(\langle z, x\rangle) \cup \mathcal{L}(\langle z, y\rangle)$,
   (c) if $\langle x, z\rangle \in E$, then set $\mathcal{L}(\langle x, z\rangle) = \mathcal{L}(\langle x, z\rangle) \cup \{\mathsf{Inv}(S) \mid S \in \mathcal{L}(\langle z, y\rangle)\}$, and
   (d) remove $\langle z, y\rangle$ from E;
2. for all nominal nodes z such that $\langle y, z\rangle \in E$
   (a) if $\{\langle x, z\rangle, \langle z, x\rangle\} \cap E = \emptyset$, then add $\langle x, z\rangle$ to E and set $\mathcal{L}(\langle x, z\rangle) = \mathcal{L}(\langle y, z\rangle)$,
   (b) if $\langle x, z\rangle \in E$, then set $\mathcal{L}(\langle x, z\rangle) = \mathcal{L}(\langle x, z\rangle) \cup \mathcal{L}(\langle y, z\rangle)$,
   (c) if $\langle z, x\rangle \in E$, then set $\mathcal{L}(\langle z, x\rangle) = \mathcal{L}(\langle z, x\rangle) \cup \{\mathsf{Inv}(S) \mid S \in \mathcal{L}(\langle y, z\rangle)\}$, and
   (d) remove $\langle y, z\rangle$ from E;
3. set $\mathcal{L}(x) = \mathcal{L}(x) \cup \mathcal{L}(y)$;
4. add $x \not\approx z$ for all z such that $y \not\approx z$; and
5. $\mathsf{Prune}(y)$.

If y was merged into x, we call x a **direct heir** of y, and we use being an **heir** of another node for the transitive closure of being a "direct heir".

Level (of Nominal Nodes) Let $o_1, \ldots, o_\ell$ be all the nominals occurring in the input concept D. We define the *level* of a node inductively as follows:

- each (nominal) node x with an $o_i \in \mathcal{L}(x)$, $1 \leq i \leq \ell$, is of level 0, and
- a nominal node x is of level i if x is not of some level $j < i$ and x has a neighbour that is of level $i - 1$.

Strategy (of Rule Application) The expansion rules in Figure 1 are applied according to the following strategy:

1. the o-rule is applied with highest priority,
2. next, the $\leqslant$- and the *NN*-rule are applied, and they are applied first to nominal nodes with lower levels (before they are applied to nodes with higher levels). In case they are both applicable to the same node, the *NN*-rule is applied first.
3. all other rules are applied with a lower priority.

We are now ready to finish the description of the tableau algorithm. A completion graph is **complete** if it contains a clash, or when none of the rules is applicable. If the expansion rules can be applied to C_0 and $\mathcal{R}$ in such a way that they yield a complete, clash-free completion graph, then the algorithm returns "C_0 is *satisfiable* w.r.t. $\mathcal{R}$", and "C_0 is *unsatisfiable* w.r.t. $\mathcal{R}$" otherwise.

Termination, Soundness, and Completeness

All but the Self–Ref-rule have been used before for fragments of $\mathcal{SROIQ}$, see Horrocks, Sattler, & Tobies (1999); Horrocks & Sattler (2002, 2004), and the three $\forall_i$-rules are the obvious counterparts to the tableau conditions (P7)–(P9).

As usual, we prove termination, soundness, and completeness of the tableau algorithm to show that it indeed decides satisfiability of $\mathcal{SROIQ}$-concepts w.r.t. Rboxes.

Theorem 16 (Termination, Soundness, and Completeness)

Let C_0 be a $\mathcal{SROIQ}$-concept in NNF and $\mathcal{R}$ a reduced Rbox.

1. *The tableau algorithm terminates when started with C_0 and $\mathcal{R}$.*
2. *The expansion rules can be applied to C_0 and $\mathcal{R}$ such that they yield a complete and clash-free completion graph if and only if there is a tableau for C_0 w.r.t. $\mathcal{R}$.*

Proof: (1): The algorithm constructs a graph that consists of a set of arbitrarily interconnected nominal nodes, and "trees" of blockable nodes with each tree rooted in r_0 or in a nominal node, and where branches of these trees might end in an edge leading to a nominal node.

Termination is a consequence of the usual $\mathcal{SHIQ}$ conditions with respect to the blockable tree parts of the graph, plus the fact that there is a bound on the number of new nominal nodes that can be added to $\mathbf{G}$ by the *NN*-rule.

The termination proof for the $\mathcal{SROIQ}$ tableaux is virtually identical to the one for $\mathcal{SHOIQ}$, whence we omit the

$\sqcap$-rule:	if	$C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$,
	then	$\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule:	if	$C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
	then	$\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{E\}$ for some $E \in \{C_1, C_2\}$
$\exists$-rule:	if	$\exists S.C \in \mathcal{L}(x)$, x is not blocked, and x has no S-neighbour y with $C \in \mathcal{L}(y)$
	then	create a new node y with $\mathcal{L}(\langle x, y\rangle) := \{S\}$ and $\mathcal{L}(y) := \{C\}$
Self–Ref-rule:	if	$\exists S.\mathsf{Self} \in \mathcal{L}(x)$ or $\mathsf{Ref}(S) \in \mathcal{R}_a$, x is not blocked, and $S \notin \mathcal{L}(\langle x, x\rangle)$
	then	add an edge $\langle x, x\rangle$ if it does not yet exist, and set $\mathcal{L}(\langle x, x\rangle) \longrightarrow \mathcal{L}(\langle x, x\rangle) \cup \{S\}$
$\forall_1$-rule:	if	$\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked, and $\forall \mathcal{B}_S.C \notin \mathcal{L}(x)$
	then	$\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{\forall \mathcal{B}_S.C\}$
$\forall_2$-rule:	if	$\forall \mathcal{B}(p).C \in \mathcal{L}(x)$, x is not indirectly blocked, $p \xrightarrow{S} q$ in $\mathcal{B}(p)$, and there is an S-neighbour y of x with $\forall \mathcal{B}(q).C \notin \mathcal{L}(y)$,
	then	$\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{\forall \mathcal{B}(q).C\}$
$\forall_3$-rule:	if	$\forall \mathcal{B}.C \in \mathcal{L}(x)$, x is not indirectly blocked, $\varepsilon \in L(\mathcal{B})$ and $C \notin \mathcal{L}(x)$
	then	$\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C\}$
choose-rule:	if	$(\leqslant n S.C) \in \mathcal{L}(x)$, x is not indirectly blocked, and there is an S-neighbour y of x with $\{C, \dot{\neg}C\} \cap \mathcal{L}(y) = \emptyset$
	then	$\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{E\}$ for some $E \in \{C, \dot{\neg}C\}$
$\geqslant$-rule:	if	1. $(\geqslant n S.C) \in \mathcal{L}(x)$, x is not blocked 2. there are not n safe S-neighbours $y_1, \ldots, y_n$ of x with $C \in \mathcal{L}(y_i)$ and $y_i \neq y_j$ for $1 \leq i < j \leq n$
	then	create n new nodes $y_1, \ldots, y_n$ with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$, $\mathcal{L}(y_i) = \{C\}$, and $y_i \neq y_j$ for $1 \leq i < j \leq n$.
$\leqslant$-rule:	if	1. $(\leqslant n S.C) \in \mathcal{L}(z)$, z is not indirectly blocked 2. $\sharp S^{\mathbf{G}}(z, C) > n$ and there are two S-neighbours x, y of z with $C \in \mathcal{L}(x) \cap \mathcal{L}(y)$, and not $x \neq y$
	then	1. if x is a nominal node then $\mathsf{Merge}(y, x)$ 2. else, if y is a nominal node or an ancestor of x then $\mathsf{Merge}(x, y)$ 3. else $\mathsf{Merge}(y, x)$
o-rule:	if	for some $o \in N_I$ there are 2 nodes x, y with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ and not $x \neq y$
	then	$\mathsf{Merge}(x, y)$
NN-rule:	if	1. $(\leqslant n S.C) \in \mathcal{L}(x)$, x is a nominal node, and there is a blockable S-neighbour y of x such that $C \in \mathcal{L}(y)$ and x is a successor of y, 2. there is no m such that $1 \leqslant m \leqslant n$, $(\leqslant m S.C) \in \mathcal{L}(x)$, and there exist m nominal S-neighbours $z_1, \ldots, z_m$ of x with $C \in \mathcal{L}(z_i)$ and $z_i \neq z_j$ for all $1 \leq i < j \leq m$.
	then	1. guess m with $1 \leqslant m \leqslant n$, and set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{(\leqslant m S.C)\}$ 2. create m new nodes $y_1, \ldots, y_m$ with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$, $\mathcal{L}(y_i) = \{C, o_i\}$, for each $o_i \in N_I$ new in $\mathbf{G}$, and $y_i \neq y_j$ for $1 \leq i < j \leq m$.

Figure 1: The Expansion Rules for the $\mathcal{SROIQ}$ Tableau Algorithm.

details and refer the reader to Horrocks & Sattler (2005). To see this, note first that the blocking technique employed for $\mathcal{SROIQ}$ is identical to the one for $\mathcal{SHOIQ}$. Next, the closure $\mathsf{fclos}(C_0, \mathcal{R})$ is defined differently, comprising concepts of the form $\forall \mathcal{B}_S(q).C$, generally yielding a size of $\mathsf{fclos}(C_0, \mathcal{R})$ that can be exponential in the depth of the role hierarchy. However, the construction of the automata can also be considered a pre-processing step and part of the input, in that case keeping the polynomial bound on the size of the closure relative to the input. Furthermore, it should be clear that the new Self–Ref-rule (only adding new reflexive edges) as well as the new clash conditions do not affect the termination of the algorithm.

(2): For the "if" direction, we can obtain a tableau $T = (\mathbf{S}, \mathcal{L}', \mathcal{E})$ from a complete and clash-free completion graph $\mathbf{G}$ by *unravelling* blockable "tree" parts of the graph as usual (these are the only parts where blocking can apply).

More precisely, paths are defined as follows. For a label blocked node x, let $b(x)$ denote a **node that blocks** x.

A **path** is a sequence of pairs of blockable nodes of $\mathbf{G}$ of the form $p = \langle (x_0, x_0'), \ldots, (x_n, x_n')\rangle$. For such a path, we define $\mathsf{Tail}(p) := x_n$ and $\mathsf{Tail}'(p) := x_n'$. With $\langle p | (x_{n+1}, x_{n+1}')\rangle$ we denote the path

$$\langle (x_0, x_0'), \ldots, (x_n, x_n'), (x_{n+1}, x_{n+1}')\rangle.$$

The set $\mathsf{Paths}(\mathbf{G})$ is defined inductively as follows:

- For each blockable node x of $\mathbf{G}$ that is a successor of a nominal node or a root node, $\langle (x, x)\rangle \in \mathsf{Paths}(\mathbf{G})$, and
- For a path $p \in \mathsf{Paths}(\mathbf{G})$ and a blockable node y in $\mathbf{G}$:
  - if y is a successor of $\mathsf{Tail}(p)$ and y is not blocked, then $\langle p | (y, y)\rangle \in \mathsf{Paths}(\mathbf{G})$, and
  - if y is a successor of $\mathsf{Tail}(p)$ and y is blocked, then $\langle p | (b(y), y)\rangle \in \mathsf{Paths}(\mathbf{G})$.

Please note that, due to the construction of Paths, all nodes occurring in a path are blockable and, for $p \in \mathsf{Paths}(\mathbf{G})$ with $p = \langle p' | (x, x')\rangle$, x is not blocked, x' is blocked iff $x \neq x'$, and x' is never indirectly blocked. Furthermore, the blocking condition implies $\mathcal{L}(x) = \mathcal{L}(x')$.

Next, we use $\mathsf{Nom}(\mathbf{G})$ for the set of nominal nodes in $\mathbf{G}$, and define a tableau $T = (\mathbf{S}, \mathcal{L}', \mathcal{E})$ from $\mathbf{G}$ as follows.

$$\mathbf{S} = \mathsf{Nom}(\mathbf{G}) \cup \mathsf{Paths}(\mathbf{G})$$

$$\mathcal{L}'(p) = \begin{cases} \mathcal{L}(\mathsf{Tail}(p)) & \text{if } p \in \mathsf{Paths}(\mathbf{G}) \\ \mathcal{L}(p) & \text{if } p \in \mathsf{Nom}(\mathbf{G}) \end{cases}$$

$$\mathcal{E}(R) = E_1 \cup E_2 \cup E_3 \cup E_4, \text{ where}$$

$$E_1 = \{\langle p, q\rangle \in \mathsf{Paths}(\mathbf{G}) \times \mathsf{Paths}(\mathbf{G}) \mid q = \langle p | (x, x')\rangle \text{ and } x' \text{ is an } R\text{-successor of } \mathsf{Tail}(p), \text{ or } p = \langle q | (x, x')\rangle \text{ and } x' \text{ is an } \mathsf{Inv}(R)\text{-successor of } \mathsf{Tail}(q)\}$$

$$E_2 = \{\langle p, x\rangle \in \mathsf{Paths}(\mathbf{G}) \times \mathsf{Nom}(\mathbf{G}) \mid x \text{ is an } R\text{-neighbour of } \mathsf{Tail}(p)\}$$

$$E_3 = \{\langle x, p\rangle \in \mathsf{Nom}(\mathbf{G}) \times \mathsf{Paths}(\mathbf{G}) \mid \mathsf{Tail}(p) \text{ is an } R\text{-neighbour of } x\}$$

$$E_4 = \{\langle x, y\rangle \in \mathsf{Nom}(\mathbf{G}) \times \mathsf{Nom}(\mathbf{G}) \mid y \text{ is an } R\text{-neighbour of } x\}$$

We already commented above on $\mathbf{S}$, and $\mathcal{L}'$ is straightforward. Unfortunately, $\mathcal{E}$ is slightly cumbersome because we must distinguish between blockable and nominal nodes.

CLAIM: T is a tableau for C_0 with respect to $\mathcal{R}$.

Firstly, by definition of the algorithm, there is an heir x_0 of r_0 with $C_0 \in \mathcal{L}(x_0)$. By the $\leqslant$-rule, x_0 is either a root node or a nominal node, and thus cannot be blocked. Hence there is some $s \in \mathbf{S}$ with $C_0 \in \mathcal{L}'(s)$. Next, we prove that T satisfies each (Pi).

- (P1), (P2), (P5) and (P6) are trivially implied by the definition of $\mathcal{L}'$ and completeness of $\mathbf{G}$.
- (P3) and (P17) follow from the construction of $\mathcal{E}$ and completeness of $\mathbf{G}$, and (P4) follows from clash-freeness.
- for (P7), consider a tuple $\langle s,t\rangle \in \mathcal{E}(R)$ with $\forall \mathcal{B}(p).C \in \mathcal{L}'(s)$ and $p \xrightarrow{R} q \in \mathcal{B}(p)$. We have to show that $\forall \mathcal{B}(q).C \in \mathcal{L}'(t)$ and distinguish four different cases:
  - if $\langle s,t\rangle \in \mathsf{Paths}(\mathbf{G}) \times \mathsf{Paths}(\mathbf{G})$, then $\forall \mathcal{B}(p).C \in \mathcal{L}(\mathsf{Tail}(s))$ and
    - either $\mathsf{Tail}'(t)$ is an R-successor of $\mathsf{Tail}(s)$. Hence completeness implies $\forall \mathcal{B}(q).C \in \mathcal{L}(\mathsf{Tail}'(t))$, and by definition of $\mathsf{Paths}(\mathbf{G})$, either $\mathsf{Tail}'(t) = \mathsf{Tail}(t)$, or $\mathsf{Tail}(t)$ blocks $\mathsf{Tail}'(t)$ and the blocking condition implies $\mathcal{L}(\mathsf{Tail}'(t)) = \mathcal{L}(\mathsf{Tail}(t))$.
    - or $\mathsf{Tail}'(s)$ is an $\mathsf{Inv}(R)$-successor of $\mathsf{Tail}(t)$. Again, either $\mathsf{Tail}'(s) = \mathsf{Tail}(s)$, or $\mathsf{Tail}(s)$ blocks $\mathsf{Tail}'(s)$ in which case the blocking condition implies that $\forall \mathcal{B}(p).C \in \mathcal{L}(\mathsf{Tail}'(s))$, and thus completeness implies that $\forall \mathcal{B}(q).C \in \mathcal{L}(\mathsf{Tail}(t))$.
  - if $\langle s,t\rangle \in \mathsf{Nom}(\mathbf{G}) \times \mathsf{Nom}(\mathbf{G})$, then $\forall \mathcal{B}(p).C \in \mathcal{L}(s)$ and t is an R-neighbour of s. Hence completeness implies $\forall \mathcal{B}(q).C \in \mathcal{L}(t)$.
  - if $\langle s,t\rangle \in \mathsf{Nom}(\mathbf{G}) \times \mathsf{Paths}(\mathbf{G})$, then $\forall \mathcal{B}(p).C \in \mathcal{L}(s)$ and $\mathsf{Tail}(t)$ is an R-neighbour of s. Hence completeness implies $\forall \mathcal{B}(q).C \in \mathcal{L}(\mathsf{Tail}(t))$.
  - if $\langle s,t\rangle \in \mathsf{Paths}(\mathbf{G}) \times \mathsf{Nom}(\mathbf{G})$, then $\forall \mathcal{B}(p).C \in \mathcal{L}(\mathsf{Tail}(s))$ and t is an R-neighbour of $\mathsf{Tail}(s)$. Hence completeness implies $\forall \mathcal{B}(q).C \in \mathcal{L}(t)$.

  In all four cases, by definition of $\mathcal{L}'$, we have $\forall \mathcal{B}(q).C \in \mathcal{L}'(t)$.
- (P8) and (P9) follow from completeness of $\mathbf{G}$.
- for (P10), consider some $s \in \mathbf{S}$ with $\exists R.C \in \mathcal{L}'(s)$.
  - If $s \in \mathsf{Paths}(\mathbf{G})$, then $\exists R.C \in \mathcal{L}(\mathsf{Tail}(s))$, $\mathsf{Tail}(s)$ is not blocked, and completeness of $\mathcal{T}$ implies the existence of an R-neighbour y of $\mathsf{Tail}(s)$ with $C \in \mathcal{L}(y)$.
    - If y is a nominal node, then $y \in \mathbf{S}$, $C \in \mathcal{L}'(y)$, and $\langle s,y\rangle \in \mathcal{E}(R)$.
    - If y is blockable and a successor of $\mathsf{Tail}(s)$, then $\langle s|(\tilde{y},y)\rangle \in \mathbf{S}$, for $\tilde{y} = y$ or $\tilde{y} = b(y)$, $C \in \mathcal{L}'(\langle s|(\tilde{y},y)\rangle)$, and $\langle s,\langle s|(\tilde{y},y)\rangle\rangle \in \mathcal{E}(R)$.
    - If y is blockable and a predecessor of $\mathsf{Tail}(s)$, then $s = \langle p|(y,y)|(\mathsf{Tail}(s),\mathsf{Tail}'(s))\rangle$, $C \in \mathcal{L}'(\langle p|(y,y)\rangle)$, and $\langle s,\langle p|(y,y)\rangle\rangle \in \mathcal{E}(R)$.
  - If $s \in \mathsf{Nom}(\mathbf{G})$, then completeness implies the existence of some R-successor x of s with $C \in \mathcal{L}(x)$.
    - If x is a nominal node, then $\langle s,x\rangle \in \mathcal{E}(R)$ and $C \in \mathcal{L}'(x)$.
    - If x is a blockable node, then x is a safe R-neighbour of s and thus not blocked. Hence there is a path $p \in \mathsf{Paths}(\mathbf{G})$ with $\mathsf{Tail}(p) = x$, $\langle s,p\rangle \in \mathcal{E}(R)$ and $C \in \mathcal{L}'(p)$.
- (P11) and (P12) are immediate consequences of the definition of "R-successor" and "R-neighbour", as well as the definition of $\mathcal{E}$.
- for (P13), consider some $s \in \mathbf{S}$ with $(\leqslant nR.C) \in \mathcal{L}'(s)$. Clash-freeness implies the existence of at most n R-neighbours y_i of s with $C \in \mathcal{L}(y_i)$. By construction, each $t \in \mathbf{S}$ with $\langle s,t\rangle \in \mathcal{E}(R)$ corresponds to an R-neighbour y_i of s or $\mathsf{Tail}(s)$, and none of these R-neighbours gives rise to more than one such y_i. Moreover, since $\mathcal{L}'(t) = \mathcal{L}(y_i)$, (P13) is satisfied.
- for (P14), consider some $s \in \mathbf{S}$ with $(\geqslant nR.C) \in \mathcal{L}'(s)$.
  - if $s \in \mathsf{Nom}(\mathbf{G})$, then completeness implies the existence of n safe R-neighbours $y_1,\ldots,y_n$ of s with and $y_j \neq y_j$, for each $i \neq j$, and $C \in \mathcal{L}(y_i)$, for each $1 \leq i \leq n$. By construction, each y_i corresponds to a $t_i \in \mathbf{S}$ with $t_i \neq t_j$, for each $i \neq j$:
    - if y_i is blockable, then it cannot be blocked since it is a safe R-neighbour of s. Hence there is a path $\langle p|(y_i,y_i)\rangle \in \mathbf{S}$ and $\langle s,\langle p|(y_i,y_i)\rangle\rangle \in \mathcal{E}(R)$.
    - if y_i is a nominal node, then $\langle s,y_i\rangle \in \mathcal{E}(R)$.
  - if $s \in \mathsf{Paths}(\mathbf{G})$, then completeness implies the existence of n R-neighbours $y_1,\ldots,y_n$ of $\mathsf{Tail}(s)$ with $y_j \neq y_j$, for each $i \neq j$, and $C \in \mathcal{L}(y_i)$, for each $1 \leq i \leq n$. By construction, each y_i corresponds to a $t_i \in \mathbf{S}$ with $t_i \neq t_j$, for each $i \neq j$:
    - if y_i is safe, then it can be blocked if it is a successor of $\mathsf{Tail}(s)$. In this case, the "pair" construction in our definition of paths ensure that, even if $b(y_i) = b(y_j)$, for some $i \neq j$, we still have $\langle p|(b(y_i),y_i)\rangle \neq \langle p|(b(y_j),b_j)\rangle$.
    - if y_i is unsafe, then $\langle s,y_i\rangle \in \mathcal{E}(R)$.

    Hence all t_i are different and, by construction, $C \in \mathcal{L}'(t_i)$, for each $1 \leq i \leq n$.
- (P15) is satisfied due to completeness of $\mathbf{G}$ and the fact that each $t \in \mathbf{S}$ with $\langle s,t\rangle \in \mathcal{E}(R)$ corresponds to an R-neighbour of s (in case $s \in \mathsf{Nom}(\mathbf{G})$) or of $\mathsf{Tail}(s)$ (in case $s \in \mathsf{Paths}(\mathbf{G})$).
- (P16) and (P18) follow from clash-freeness and definition of $\mathcal{E}$, (P19) follows trivially from the initialisation of $\mathbf{G}$, and (P20) is due to completeness of $\mathbf{G}$ and the fact that nominal nodes are not "unravelled".

For the "only if" direction, given a tableau $T = (\mathbf{S},\mathcal{L}',\mathcal{E})$ for C_0 w.r.t. $\mathcal{R}$, we can apply the non-deterministic rules, i.e., the $\sqcup$-, *choose*-, $\leqslant$-, and *NN*-rule, in such a way that we obtain a complete and clash-free graph: inductively with the generation of new nodes, we define a mapping π from nodes in the completion graph to individuals in $\mathbf{S}$ of the tableau in such a way that,

1. for each node x, $\mathcal{L}(x) \subseteq \mathcal{L}'(\pi(x))$,
2. for each pair of nodes x, y and each role R, if y is an R-successor of x, then $\langle \pi(x), \pi(y) \rangle \in \mathcal{E}(R)$, and
3. $x \neq y$ implies $\pi(x) \neq \pi(y)$.

This is analogous to the proof in Horrocks, Sattler, & Tobies (1999) with the additional observation that, due to (P20), application of the o-rule does not lead to a clash of the form (7) as given in Definition 15. Similarly, an application of the Self–Ref-rule does not lead to a clash of the form (3) due to Condition (P4), a clash of the form (4) can not occur due to (P16), and a clash of the form (5) can not occur due to (P18).

□

From Theorems 9, 14 and 16, we thus arrive at the following theorem:

Theorem 17 (Decidability) *The tableau algorithm decides satisfiability and subsumption of $\mathcal{SROIQ}$-concepts with respect to Aboxes, Rboxes, and Tboxes.*

Outlook and Future Work

We introduced a description logic, $\mathcal{SROIQ}$, that overcomes certain shortcomings in expressiveness of other DLs. We have used $\mathcal{SHOIQ}$ and $\mathcal{RIQ}$ as a starting point, extended them with "useful-yet-harmless" expressive means, and extended the tableau algorithm accordingly. $\mathcal{SROIQ}$ is intended to be a basis for future extensions of OWL, and has already been adopted as the logical basis of OWL 1.1.

It is left for future work to determine whether the restrictions to simple roles can be relaxed, to pinpoint the exact computational complexity of $\mathcal{SROIQ}$, and to include further role assertions such as the more general version of antisymmetry to allow a better modeling of mereological notions (Goodwin, 2005).

A further line of investigation concerns concrete datatypes with inverse functional datatype properties: these are of interest since they allow to express simple key constraints. For instance, we might want to use a datatype property `SSN` for social security number as a key for US citizen.

References

Areces, C.; Blackburn, P.; Hernandez, B.; and Marx, M. 2003. Handling Boolean Aboxes. In *Proc. of DL 2003*.

Baader, F., and Hanschke, P. 1991. A Schema for Integrating Concrete Domains into Concept Languages. In *Proc. of IJCAI-12*, 452–457.

Baader, F.; Bürckert, H.-J.; Nebel, B.; Nutt, W.; and Smolka, G. 1993. On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations. *Journal of Logic, Language and Information* 2:1–18.

Baader, F.; Lutz, C.; Milicic, M.; Sattler, U.; and Wolter, F. 2005. Integrating Description Logics and Action Formalisms: First Results. In *Proc. of AAAI-20*. AAAI Press.

Baader, F. 1991. Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles. In *Proc. of IJCAI-12*.

Blackburn, P., and Seligman, J. 1995. Hybrid languages. *J. of Logic, Language and Information* 4:251–272.

Casati, R., and Varzi, A. 1999. *Parts and Places. The Structure of Spatial Representation*. Cambridge, MA: MIT Press.

Goodwin, J. 2005. Experiences of Using OWL at the Ordnance Survey. In *Proc. of OWL: Experiences and Directions*.

Horrocks, I., and Sattler, U. 2001. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of IJCAI-17*, 199–204.

Horrocks, I., and Sattler, U. 2002. Optimised reasoning for $\mathcal{SHIQ}$. In *Proc. of ECAI-15*.

Horrocks, I., and Sattler, U. 2004. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. *Artificial Intelligence* 160:79–104.

Horrocks, I., and Sattler, U. 2005. A Tableaux Decision Procedure for $\mathcal{SHOIQ}$. In *Proc. of IJCAI-19*. Morgan Kaufmann, Los Altos.

Horrocks, I.; Kutz, O.; and Sattler, U. 2005. The Irresistible $\mathcal{SRIQ}$. In *Proc. of OWL: Experiences and Directions*.

Horrocks, I.; Patel-Schneider, P. F.; and van Harmelen, F. 2003. From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language. *J. of Web Semantics* 1(1):7–26.

Horrocks, I.; Sattler, U.; and Tobies, S. 1999. Practical Reasoning for Expressive Description Logics. In Ganzinger, H.; McAllester, D.; and Voronkov, A., eds., *Proc. of LPAR-6*, volume 1705 of *LNAI*, 161–180. Springer.

Horrocks, I.; Sattler, U.; and Tobies, S. 2000. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In MacAllester, D., ed., *Proc. of CADE-17*, volume 1831 of *LNCS*. Germany: Springer.

Pan, J., and Horrocks, I. 2003. Web ontology reasoning with datatype groups. In Fensel, D.; Sycara, K.; and Mylopoulos, J., eds., *Proc. of ISWC 2003*, number 2870 in LNCS, 47–63. Springer.

Schaerf, A. 1994. Reasoning with individuals in concept languages. *Data and Knowledge Engineering* 13(2):141–176.

Schild, K. 1991. A Correspondence Theory for Terminological Logics: Preliminary Report. In *Proc. of IJCAI-12*, 466–471.

Simons, P. 1987. *Parts: A Study in Ontology*. Oxford: Clarendon Press.

Wolstencroft, K.; Brass, A.; Horrocks, I.; Lord, P.; Sattler, U.; Turi, D.; and Stevens, R. 2005. A Little Semantic Web Goes a Long Way in Biology. In *Proc. of ISWC-4*, number 3729 in LNCS, 786–800. Springer.

Appendix M:

FaCT++ source code as of 21 April 2006. Enclosed with the full electronic version of this report as the file FaCT++src.tar.